09/781,284 #4

日本国特許庁
PATENT OFFICE
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されてる事項と同一であることを証明する。
This is to certify that the annexed is a true copy of the following application as filed this Office.

出願年月日
Date of Application: ２０００年　５月　８日

出願番号
Application Number: 特願２０００－１３５０１０

出願人
Applicant (s): 株式会社東芝

２００１年　１月１９日

特許庁長官
Commissioner,
Patent Office 及川耕造

出証番号　出証特２０００－３１１２７４８

【書類名】　特許願

【整理番号】　13B004035

【提出日】　平成12年 5月 8日

【あて先】　特許庁長官殿

【国際特許分類】　H04L 9/00

G06F 9/00

【発明の名称】　マイクロプロセッサ、これを用いたマルチタスク実行方法、およびマルチレッド実行方法

【請求項の数】　15

【発明者】

【住所又は居所】　神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内

【氏名】　橋本　幹生

【発明者】

【住所又は居所】　神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内

【氏名】　藤本　謙作

【発明者】

【住所又は居所】　神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内

【氏名】　白川　健治

【発明者】

【住所又は居所】　神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内

【氏名】　寺本　圭一

【特許出願人】

【識別番号】　000003078

【氏名又は名称】　株式会社　東芝

【代理人】

【識別番号】　100083806

【弁理士】

【氏名又は名称】　三好　秀和

【電話番号】　03-3504-3075

【選任した代理人】

【識別番号】　100068342

【弁理士】

【氏名又は名称】　三好　保男

【選任した代理人】

【識別番号】　100100712

【弁理士】

【氏名又は名称】　岩▲崎▼　幸邦

【選任した代理人】

【識別番号】　100100929

【弁理士】

【氏名又は名称】　川又　澄雄

【選任した代理人】

【識別番号】　100108707

【弁理士】

【氏名又は名称】　中村　友之

【選任した代理人】

【識別番号】　100095500

【弁理士】

【氏名又は名称】　伊藤　正和

【選任した代理人】

【識別番号】　100101247

【弁理士】

【氏名又は名称】　高橋　俊一

【選任した代理人】

【識別番号】　100098327

【弁理士】

【氏名又は名称】　高松　俊雄

【手数料の表示】

【予納台帳番号】　001982

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　マイクロプロセッサ、これを用いたマルチタスク実行方法、およびマルチレッド実行方法

【特許請求の範囲】

【請求項１】　プログラムごとに異なる暗号化鍵により暗号化された複数のプログラムをマイクロプロセッサ外部の記憶手段から読み出す読み出し手段と、

前記読み出し手段にて読み出した前記複数のプログラムを、それぞれ対応する復号化鍵で復号化する復号化手段と、

前記復号化された複数のプログラムを実行する実行手段と、

前記複数のプログラムのうち、一のプログラムの実行を中断する場合に、前記一のプログラムの実行状態を示す情報と、前記一のプログラムの暗号化鍵とを、前記マイクロプロセッサに固有の暗号化鍵で暗号化し、この暗号化した情報をコンテキスト情報として前記マイクロプロセッサ外部の記憶手段に書き込む実行状態書き込み手段と、

前記一のプログラムを再開する場合に、前記マイクロプロセッサに固有の暗号化鍵に対応する固有の復号化鍵で前記コンテキスト情報を復号化し、復号化された再開予定のプログラムの暗号化鍵が、前記一のプログラム本来の暗号化鍵と一致した場合にのみ、前記一のプログラムの実行を再開する再開手段と、

を具備することを特徴とする１チップまたは１パッケージのマイクロプロセッサ。

【請求項２】　外部へ読み出すことのできない前記マイクロプロセッサ内部の記憶手段と、

前記複数のプログラムの処理対象であるデータのための暗号化属性を、前記マイクロプロセッサ内部の記憶手段に書き込む暗号化属性書き込み手段と、

前記複数のプログラムの処理対象であるデータを、前記暗号化属性に基づいて暗号化するデータ暗号化手段と、

をさらに具備し、前記暗号化属性の少なくとも一部は、前記コンテキスト情報に含まれることを特徴とする請求項１の１チップまたは１パッケージのマイクロプロセッサ。

【請求項３】　前記実行状態書き込み手段は、マイクロプロセッサに固有の秘密情報に基づく署名をコンテキスト情報に付与し、

前記再開手段は、復号化されたコンテキスト情報に含まれる前記署名が、前記マイクロプロセッサに固有の秘密情報に基づく署名と一致する場合にのみ、前記一のプログラムを再開することを特徴とする請求項１に記載の１チップまたは１パッケージのマイクロプロセッサ。

【請求項４】　外部へ読み出すことのできない固有の秘密鍵を内部に保持した１チップまたは１パッケージのマイクロプロセッサであって、

あらかじめ前記秘密鍵に対応する公開鍵によって暗号化されたコード暗号化鍵をマイクロプロセッサ外部の記憶手段から読み出す第１の読み出し手段と、

前記第１の読み出し手段で読み出した前記コード暗号化鍵を、前記秘密鍵を用いて復号化する第１の復号化手段と、

プログラムごとに異なる前記コード暗号化鍵によって暗号化された複数のプログラムを、前記マイクロプロセッサ外部の記憶手段から読み出す第２の読み出し手段と、

前記第２の読み出し手段にて読み出した前記複数のプログラムを復号化する第２の復号化手段と、

前記復号化された複数のプログラムを実行する実行手段と、

前記複数のプログラムのうち、一のプログラムの実行を中断する場合に、前記一のプログラムの実行状態を示す情報と、前記プログラムのコード暗号化鍵とを前記公開鍵によって暗号化し、暗号化した情報をコンテキスト情報として、前記マイクロプロセッサ外部の記憶手段に書き込む実行状態書き込み手段と、

前記一のプログラムの実行を再開する場合に、前記マイクロプロセッサ外部の記憶手段から前記コンテキスト情報を読み出し、前記秘密鍵でコンテキスト情報を復号化し、復号化されたコンテキスト情報に含まれるコード暗号化鍵が、前記一のプログラムの本来のコード暗号化鍵と一致する場合にのみ、前記一のプログラムの実行を再開する再開手段と、

を具備することを特徴とするマイクロプロセッサ。

【請求項５】　外部へ読み出すことのできない固有の秘密鍵を内部に保持す

る１チップまたは１パッケージのマイクロプロセッサであって、

あらかじめ前記秘密鍵に対応した公開鍵によって暗号化されたコード暗号化鍵をマイクロプロセッサ外部の記憶手段から読み出す第１の読み出し手段と、

前記第１の読み出し手段で読み出した前記コード暗号化鍵を、前記秘密鍵を用いて復号化する第１の復号化手段と、

それぞれが異なる前記コード暗号化鍵で暗号化された複数のプログラムを、前記マイクロプロセッサ外部の記憶手段から読み出す第２の読み出し手段と、

前記第２の読み出し手段によって読み出された複数のプログラムを復号化する第２の復号化手段と、

前記復号化された複数のプログラムを実行する実行手段と、

前記複数のプログラムのうち一のプログラムの実行を中断する場合に、一時鍵として乱数を発生し、前記一のプログラムの実行状態を示す情報を前記一時鍵で暗号化した第１の値と、前記一時鍵を前記一のプログラムのコード暗号化鍵で暗号化した第２の値と、前記一時鍵を前記マイクロプロセッサの秘密鍵で暗号化した第３の値とをコンテキスト情報として前記マイクロプロセッサ外部の記憶手段に書き込む実行状態書き込み手段と、

前記一のプログラムの実行を再開する場合に、前記マイクロプロセッサ外部の記憶手段から前記コンテキスト情報を読み出し、前記秘密鍵を用いて、前記コンテキスト情報に含まれる第３の値としての一時鍵を復号化し、復号化された一時鍵を用いて前記コンテキスト情報に含まれる実行状態情報を復号化するとともに、再開予定のプログラムのコード暗号化鍵を用いて、前記コンテキスト情報に含まれる第２の値としての一時鍵を復号化し、前記第２の値を復号化した一時鍵が、前記第３の値を秘密鍵で復号化した一時鍵と一致する場合にのみ前記一のプログラムの実行を再開する再開手段と、

を具備することを特徴とするマイクロプロセッサ。

【請求項６】　前記実行状態書き込み手段は、前記マイクロプロセッサに固有の秘密鍵による署名を前記コンテキスト情報に付与し、

前記再開手段は、復号化された前記コンテキスト情報に含まれる前記署名が、前記マイクロプロセッサに固有の秘密鍵による本来の署名と一致する場合にのみ

、前記一のプログラムを再開することを特徴とする請求項５に記載の１チップまたは１パッケージのマイクロプロセッサ。

【請求項７】　外部へ読み出すことのできない固有の秘密鍵を内部に保持した１チップまたは１パッケージのマイクロプロセッサであって、

あらかじめ前記秘密鍵に対応する公開鍵によって暗号化された暗号化鍵をマイクロプロセッサ外部の記憶手段から読み出す第１の読み出し手段と、

前記第１の読み出し手段で読み出した前記暗号化鍵を、前記秘密鍵を用いて復号化する第１の復号化手段と、

プログラムごとに異なる前記暗号化鍵によって暗号化された複数のプログラムを、前記マイクロプロセッサ外部の記憶手段から読み出す第２の読み出し手段と、

前記第２の読み出し手段にて読み出した前記複数のプログラムを復号化する第２の復号化手段と、

前記復号化された複数のプログラムを実行する実行手段と、

外部へ読み出すことのできないマイクロプロセッサ内部の記憶手段と、

前記プログラムが処理するデータのための暗号化属性を、前記マイクロプロセッサ内部の記憶手段に書き込む暗号化属性書き込み手段と、

前記プログラムが処理するデータを、前記マイクロプロセッサ内部の記憶手段に書き込まれた暗号化属性に基づいて暗号化するデータ暗号化手段と、

前記複数のプログラムのうち、一のプログラムの実行を中断する場合に、前記一のプログラムの実行状態を示し、かつ前記暗号化属性の少なくとも一部を含む実行状態情報と、前記一のプログラムの暗号化鍵とを、前記公開鍵によって暗号化し、暗号化した情報をコンテキスト情報として、前記マイクロプロセッサ外部の記憶手段に書き込む実行状態書き込み手段と、

前記一のプログラムの実行を再開する場合に、前記秘密鍵でコンテキスト情報を復号化し、復号化されたコンテキスト情報に含まれる暗号化鍵が、前記一のプログラムの本来の暗号化鍵と一致する場合にのみ、前記一のプログラムの実行を再開する再開手段と、

を具備することを特徴とするマイクロプロセッサ。

【請求項８】　外部へ読み出すことのできない固有の秘密鍵を内部に保持した１チップまたは１パッケージのマイクロプロセッサであって、

あらかじめ前記秘密鍵に対応する公開鍵によって暗号化された暗号化鍵をマイクロプロセッサ外部の記憶手段から読み出す第１の読み出し手段と、

前記第１の読み出し手段で読み出した前記暗号化鍵を、前記秘密鍵を用いて復号化する第１の復号化手段と、

プログラムごとに異なる前記暗号化鍵によって暗号化された複数のプログラムを、前記マイクロプロセッサ外部の記憶手段から読み出す第２の読み出し手段と、

前記第２の読み出し手段にて読み出した前記複数のプログラムを復号化する第２の復号化手段と、

前記復号化された複数のプログラムを実行する実行手段と、

前記プログラムから参照されるデータのための暗号化属性と暗号化属性特定情報とが格納された、外部へ読み出すことのできないマイクロプロセッサ内部の記憶手段と、

前記暗号化属性特定情報に関連し、前記マイクロプロセッサに固有の署名を含む関連情報を、前記マイクロプロセッサ外部の記憶手段に書き込む関連情報書き込み手段と、

プログラムが参照するデータのアドレスに基づいて、前記マイクロプロセッサ外部の記憶手段から、前記関連情報を読み込む関連情報読み込み手段と、

前記読み込み手段により読み込まれた関連情報に含まれる署名を、前記秘密鍵によって検証し、マイクロプロセッサ固有の署名と一致した場合にのみ、前記暗号化属性特定情報および前記読み込まれた関連情報に基づいて、データ参照のための暗号化鍵およびアルゴリズムを決定し、前記プログラムによるデータの参照を許可する許可手段と、

前記プログラムから参照されるデータを、前記マイクロプロセッサ内部の記憶手段に記憶された前記暗号化属性に基づいて暗号化するデータ暗号化手段と、

前記複数のプログラムのうち、一のプログラムの実行を中断する場合に、前記一のプログラムの実行状態を示し、かつ前記暗号化属性の少なくとも一部が含ま

れた実行状態情報と、前記一のプログラムの暗号化鍵とを、前記公開鍵によって暗号化し、暗号化した情報をコンテキスト情報として、前記マイクロプロセッサ外部の記憶手段に書き込む実行状態書き込み手段と、

前記一のプログラムの実行を再開する場合に、前記秘密鍵でコンテキスト情報を復号化し、復号化された再開予定のプログラムの暗号化鍵が、前記一のプログラムの本来の暗号化鍵と一致する場合にのみ、前記一のプログラムの実行を再開する再開手段と、

を具備することを特徴とするマイクロプロセッサ。

【請求項９】　前記関連情報は、前記暗号化属性特定情報と、マイクロプロセッサ固有の秘密情報と、前記関連情報のうちプロセッサ固有の署名を除く部分とに基づいて生成される第２の署名を含むことを特徴とする請求項８に記載のマイクロプロセッサ。

【請求項１０】　前記実行手段で実行されるプログラムが変更され、前記プログラムの暗号化鍵が変更された場合は、前記マイクロプロセッサ内部の記憶手段に記録された、前記プログラムから参照されるデータの暗号化属性を初期化することを特徴とする請求項７に記載のマイクロプロセッサ。

【請求項１１】　プログラムごとに異なる暗号化鍵により暗号化された複数のプログラムを、マイクロプロセッサ外部の記憶手段から読み出す読み出し手段と、

前記読み出し手段により読み出された前記複数のプログラムを復号化する復号化手段と、

前記復号化された複数のプログラムを実行する手段と、

第１の命令の実行に基づく、前記複数のプログラムの中の一のプログラムの実行中に、この実行状態を示す情報と、前記一のプログラムの暗号化鍵とを、コンテキスト情報として前記マイクロプロセッサ外部の記憶手段に書き込む書き込み手段と、

第２の命令の実行に基づいて、前記外部の記憶手段に書き込まれたコンテキスト情報を読み出す読み出し手段と、

前記読み出されたコンテキスト情報に含まれる前記一のプログラムの暗号化鍵

と、前記第２の命令の実行に基づき実行しているプログラムの暗号化鍵とを比較し、一致した場合にのみ、前記コンテキスト情報に含まれる実行状態を復元する復元手段と、

を具備することを特徴とする１チップまたは１パッケージのマイクロプロセッサ。

【請求項１２】　それぞれ異なる暗号化鍵により暗号化された複数のプログラムのうち、一のプログラムを、マイクロプロセッサ外部の記憶手段から読み出すステップと、

前記読み出したプログラムを、このプログラムに対応する復号化鍵で復号化し、実行するステップと、

前記一のプログラムの実行が中断された場合に、前記一のプログラムの実行状態を示す情報と、前記一のプログラムの暗号化鍵とを、前記マイクロプロセッサに固有の暗号化鍵で暗号化し、この暗号化した情報をコンテキスト情報として前記マイクロプロセッサ外部の記憶手段に書き込むステップと、

前記コンテキスト情報を、前記マイクロプロセッサに固有の暗号化鍵に対応するマイクロプロセッサに固有の復号化鍵で復号化するステップと、

復号化された再開予定のプログラムの暗号化鍵と、前記一のプログラム本来の暗号化鍵とを比較し、一致した場合にのみ、前記一のプログラムの実行を再開するステップと、

を含むマルチタスク実行方法。

【請求項１３】　あらかじめ異なるコード暗号化鍵で暗号化された複数のプログラムと、前記コード暗号化鍵の各々をあらかじめマイクロプロセッサに固有の公開鍵で暗号化したコード暗号化鍵とを、前記マイクロプロセッサ外部の記憶手段に格納するステップと、

前記公開鍵で暗号化されたコード暗号化鍵を、前記マイクロプロセッサ外部の記憶手段から読み出し、前記公開鍵に対応するマイクロプロセッサに固有の秘密鍵で復号化するステップと、

前記複数のプログラムを、前記マイクロプロセッサ外部の記憶手段から読み出して、前記復号化されたコード暗号化鍵で復号化するステップと、

前記復号化された複数のプログラムを実行するステップと、

前記複数のプログラムのうち、一のプログラムの実行が中断される場合に、当該一のプログラムの実行状態を示す情報と、このプログラムのコード暗号化鍵とを、前記公開鍵によって暗号化し、これをコンテキスト情報として前記マイクロプロセッサ外部の記憶手段に書き込むステップと、

前記マイクロプロセッサ外部の記憶手段から前記コンテキスト情報を読み出し、前記秘密鍵でコンテキスト情報を復号化するステップと、

前記復号化されたコンテキスト情報に含まれるコード暗号化鍵と、前記一のプログラムの本来のコード暗号化鍵とを比較し、一致する場合にのみ、前記一のプログラムの実行を再開するステップと、

を含むマルチタスク実行方法。

【請求項１４】　あらかじめ異なる暗号化鍵で暗号化された複数のプログラムと、前記暗号化鍵の各々をあらかじめマイクロプロセッサに固有の公開鍵で暗号化した暗号化鍵とを、前記マイクロプロセッサ外部の記憶手段に格納するステップと、

前記複数のプログラムの各々の処理対象となるデータのための暗号化属性を、外部へ読み出すことのできないマイクロプロセッサ内部の記憶手段に書き込むステップと、

前記公開鍵で暗号化された暗号化鍵を、前記マイクロプロセッサ外部の記憶手段から読み出し、前記公開鍵に対応するマイクロプロセッサに固有の秘密鍵で復号化するステップと、

前記暗号化された複数のプログラムを、前記マイクロプロセッサ外部の記憶手段から読み出し、前記復号化された暗号化鍵で復号化するステップと、

前記復号化された複数のプログラムを実行する実行するステップと、

前記複数のプログラムのうち、一のプログラムの実行を中断する場合に、前記一のプログラムの実行状態を示し、かつ前記暗号化属性の少なくとも一部を含む実行状態情報と、前記一のプログラムのコード暗号化鍵とを、前記公開鍵によって暗号化し、これをコンテキスト情報として、前記マイクロプロセッサ外部の記憶手段に書き込むステップと、

前記プログラムの処理対象であるデータを、前記マイクロプロセッサ内部の記憶手段に書き込まれた暗号化属性に基づいて暗号化し、暗号化されたデータを前記マイクロプロセッサ外部の記憶手段に保存するステップと、

前記コンテキスト情報を前記マイクロプロセッサの秘密鍵で復号化するステップと、

前記復号化されたコンテキスト情報に含まれる暗号化鍵が、前記一のプログラムの本来の暗号化鍵とを比較し、一致する場合にのみ、前記一のプログラムの実行を再開するステップと、

を含むマルチタスク実行方法。

【請求項１５】　それぞれ異なる暗号化鍵であらかじめ暗号化された複数のプログラムを、マイクロプロセッサ外部の記憶手段に格納するステップと、

前記複数のプログラムを前記マイクロプロセッサ外部の記憶手段から読み出して、復号化してから実行するステップと、

第１の命令の実行に基づき、前記複数のプログラムの中の一のプログラムの実行中に、この実行状態を示す情報と、前記一のプログラムの暗号化鍵とを、コンテキスト情報として前記マイクロプロセッサ外部の記憶手段に書き込むステップと、

第２の命令の実行に基づいて、前記マイクロプロセッサ外部の記憶手段に書き込まれたコンテキスト情報を読み出すステップと、

前記読み出されたコンテキスト情報に含まれる前記一のプログラムの暗号化鍵と、前記所定の第２の命令を実行中のプログラムの暗号化鍵とを比較し、一致した場合にのみ、前記コンテキスト情報に含まれる実行状態を復元するステップと、

を含むマルチスレッド実行方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、マルチタスクのプログラム実行環境下で、実行コードや処理対象であるデータの不正な改変を防止することのできるマイクロプロセッサ、これを用

いたマルチタスク実行方法、およびマルチレッド実行方法に関する。

【０００２】

【従来の技術】

近年マイクロプロセッサの性能向上は著しく、従来の用途である計算やグラフィックに留まらず動画像、音声の再生や編集加工が可能となっている。これによりマイクロプロセッサを応用したエンドユーザ向けシステム（以下ＰＣと呼ぶ）でユーザはさまざまな動画像や音声を楽しめるようになった。ＰＣを動画像、音声の再生装置として使うことは、ＰＣが持っている計算能力と組み合わせてゲームなどの応用ができることは当然として、固定的なハードウェアを必要としないため、すでにＰＣを持っているユーザにとっては安価に動画像、音声の再生が楽しめるという利点もある。

【０００３】

しかしながらＰＣで画像／音声を扱う場合に問題となるのがその画像／音声の著作権の保護である。ＭＤやデジタルビデオデッキでは、不正なコピーを防ぐための機構がそれら装置に組み込まれることによって無制限なコピーが防止されていた。そこでは、装置を改造して不正コピーを行うことは極めて稀であり、また仮に改造された装置があったとしてもそれが広く流通することは充分防止できるという仮定がなされていた。実際、世界的に見て不正コピーを目的とする装置の製造、販売は法律によって禁止される方向にあり、不正コピーによる被害は著作権者の収益を大きく脅かすほどの問題とはなっていない。

【０００４】

ところが、ＰＣ上で画像、音楽の再生を行うときにデータを扱うのではソフトウェアである。ＰＣではエンドユーザがソフトウェアの改変を自由に行うことができてしまう。ソフトウェアの場合、ある程度の知識があればプログラムを解析することにより、それらソフトウェアを改造して不正コピーをすることが可能であることは充分想像できる。さらに問題なのはそうして作られて不正コピーのソフトウェアはハードウェアと比べて、ネットワークなどの媒体を通じて広まりやすいことがある。

【０００５】

この問題を解決するために、著作権が問題となる商業映画や音楽の再生に使われるＰＣソフトウェアは、ソフトウェアを暗号化するなどの手法により、解読、改竄されることを防止する技術が使われている。この技術は耐タンパソフトウェア技術と呼ばれている（David Aucsmith et.al; “Tamper Resistant Software: An Implementation”, Proceeding of the 1996 Intel Software Developer's Conference）。

【０００６】

また、動画像、音声に限らずＰＣを通じてユーザに提供される著作物あるいはノウハウとして価値のある情報を不正にコピーされたり、ＰＣソフトウェア自体に含まれるノウハウなどを解析から守るのにも耐タンパソフトウェア技術は有効である。

【０００７】

しかし、耐タンパソフトウェア技術は、プログラムのうち保護を要する部分を実行開始時には暗号化しておき、その部分を実行する前に復号化し、実行終了後に再び暗号化することにより、逆アセンブラ、デバッガなどの解析ツールによる解析を困難にするという技術である。したがって、プログラムがプロセッサによって実行可能である以上、プログラムの開始時から順を追って解析していけば必ず解析することが可能になる。

【０００８】

これは、著作権者がＰＣを利用した動画像、音声を再生するシステムに著作物を提供する際の妨げとなっている。

【０００９】

また、その他の耐タンパソフトの応用についても同様の弱点があり、ＰＣを通じた高度な情報サービスや企業、個人のノウハウを含んだプログラムのＰＣへの適用の妨げにもなっている。

【００１０】

上記のことはソフトウェアの保護一般にあてはまる問題であるが、これに加えて、オープンプラットフォームであるＰＣには、システムのソフトウェア的土台となるはずのオペレーションシステム（以下、適宜「ＯＳ」と略する）自体が改

変されて攻撃の手段とされるという問題もある。技術を持ち、悪意のあるユーザは、自分が所有するＰＣのＯＳに改造を加え、ＯＳの持つ特権を利用して、ＯＳにアプリケーションプログラムに埋め込まれている著作権保護機構を無効化させたり、その機構を解析させたりできるからである。

【００１１】

現代のＯＳは、ＣＰＵに備えられた実行制御機能やメモリへの特権操作機能を利用して、コンピュータの制御下にある資源の管理や利用の調停を行なっている。管理の対象として、従来からの対象であるデバイスや、ＣＰＵ、メモリ資源に加え、ネットワークやアプリケーションレベルのＱｏＳ（Quality of Service）などが加わってきている。とはいえ、資源管理の基本はあくまでも、プログラムの実行に必要な資源の割り当てである。すなわち、そのプログラムの実行に対するＣＰＵ時間の割り当てと、実行に必要なメモリ空間の割り当てが、資源管理の基本となる。それ以外のデバイス、ネットワーク、アプリケーションＱｏＳの制御は、これらの資源に対してアクセスするプログラムの実行を制御することによって（ＣＰＵ時間とメモリを割り当てることによって）行われる。

【００１２】

ＯＳは、ＣＰＵ時間の割り当てと、メモリの割り当てを行うための特権を有する。すなわち、ＣＰＵ時間の割り当てをするために、アプリケーションプログラムを任意の時点で停止、再開する特権と、任意の時点でアプリケーションに割り当てたメモリ空間の内容を、異なる階層のメモリに移動する特権である。後者の特権は、異なるアクセス速度と容量を持つ（通常は）階層化されたメモリシステムをアプリケーションから隠ぺいして、フラットなメモリ空間をアプリケーションに提供するためにも使用される。

【００１３】

この２つの特権を持つことにより、任意の時点でＯＳはアプリケーションの実行状態を停止してスナップショットをとり、それをコピーまたは書き換えて再開することが可能である。この機能はアプリケーションのもつ秘密を解析する道具にも使うことができる。

【００１４】

コンピュータ上のアプリケーションに対する解析を防ぐために、プログラムやデータの暗号化を行う技術はすでにいくつか知られている（Hampsonによる米国特許第4,47,902号、Hartmanによる米国特許第5,224,166号、Davisによる米国特許第5,806,706号、Takahasiによる米国特許第5,825,878号、Leonard et.alによる米国特許第6,003,117号、特開平１１－２８２７５６号公報等）。しかし、これらの技術では、上述したようなオペレーションシステムの特権動作からプログラムの動作とデータの秘密を守ることは考慮されていない。

【００１５】

Intel社のｘ８６アーキテクチャに基づく従来技術（Hartmanの米国特許第5,224,166号、“System for seamless processing of encrypted and non-encrypted data and instructions”、以下、「従来技術１」とする）は、実行コードとデータを、予め定められた暗号化鍵Ｋｘで暗号化してメモリに格納する技術を開示している。Ｋｘは、プロセッサに埋め込まれた秘密鍵Ｋｓに対応する公開鍵Ｋｐで暗号化されて$E_{Kr}[K_x]$の形で与えられる。したがって、Ｋｓを知っているプロセッサのみがメモリ上の暗号化された実行コードを復号化できる。暗号化鍵Ｋｘは、セグメントレジスタと呼ばれるプロセッサ内部のレジスタに格納される。

【００１６】

この機構により、確かにプログラムコードを暗号化することでユーザからコードの秘密をある程度保護することは可能である。また、コードの暗号化鍵Ｋｘを知らない者がコードを意図した通りに改変することや、暗号化鍵Ｋｘで復号化されて実行可能なコードを新たに作成することは暗号学的に困難となる。

【００１７】

【発明が解決しようとする課題】

しかし、この技術を適用したシステムでは、実行コードの暗号化を解かなくても、コンテキスト切替と呼ばれるＯＳが持つ特権を利用することで、プログラムの解析が可能になるという欠点がある。

【００１８】

具体的には、割り込みによってプログラムの実行が中断された時や、プログラ

ムがシステムコール呼び出しのために自発的にソフトウェア割り込み命令を呼び出した時、ＯＳは他のプログラム実行のために、コンテキストの切替処理を行う。コンテキストの切替とは、その時点でのレジスタ値の集合からなるプログラムの実行状態（以下コンテキスト情報と呼ぶ）をメモリに保存し、あらかじめメモリに保存されていた別のプログラムのコンテキスト情報をレジスタに復帰させる操作である。ｘ８６プロセッサにおける従来のコンテキスト保存の形式を、図１８に示す。ここにはアプリケーションが使うレジスタの内容が全て含まれている。中断されたプログラムのコンテキスト情報は、プログラムが再開される時に元のレジスタに復帰される。コンテキスト切替は複数のプログラムを並行動作させる上で不可欠の機能である。従来の技術では、ＯＳはコンテキスト切替の時にレジスタ値を読むことができるので、そのプログラムの実行状態がどのように変化したかに基づいて、全てとは限らないが、プログラムの行なっている動作のほとんどの部分を推定することができる。

【００１９】

さらにタイマなどの設定によって例外を発生させるタイミングを制御すれば、プログラムの任意の実行時点でこの処理を行うことができる。また、実行を中断、解析するばかりでなく、レジスタの情報を悪意で書き換えることも可能である。レジスタの書き換えは、プログラムの動作を変更できるばかりでなく、プログラムの解析も容易にする。ＯＳはアプリケーションの任意の状態を保存することができるため、ある時点で保存した状態から、繰り返し何度もレジスタ値を書き換えてプログラムを動作させてプログラムの動作を解析することが可能なのである。上記の機能に加えてプロセッサにはステップ実行などのデバッグ支援機構があり、ＯＳはこれらの機能を全て利用してアプリケーションの解析を行うことができてしまうという失点がある。

【００２０】

また、前記従来技術１ではデータに関しては、暗号化されたコードセグメントを通じたプログラム実行によってのみ、プログラムは暗号化データにアクセスできるとされている。このとき、それぞれ異なる暗号化鍵で暗号化されたプログラムがあっても、暗号化されたデータは、プログラムがどのような暗号化鍵で暗号

化されているかにかかわらず、任意の鍵を用いて、暗号化されたプログラムから自由に読むことができるという問題がある。ここに示した従来技術では、ＯＳとアプリケーションが独立に秘密を持ち、アプリケーションの秘密をＯＳから守ることや、複数のプログラム供給者がそれぞれに秘密を持つことは考えられていないのである。

【００２１】

もちろん、仮想記憶機構に設けられた保護機能によって、アプリケーションの間でメモリ空間を隔離したり、アプリケーションによるシステムメモリへのアクセスを禁止することは既存のプロセッサでも可能である。しかし、仮想記憶機構がＯＳの管理下にある以上、アプリケーションの秘密を守ることをＯＳの管理下ある機能に依存することはできない。ＯＳは保護機構を無視してデータをアクセスすることができるからであり、前述のように仮想記憶機能の提供においてその特権は不可欠である。

【００２２】

別の従来技術として、特開平１１－２８２７５６（以下従来技術２と呼ぶ）では、アプリケーションの持つ秘密情報を保存するためにＣＰＵ内に設けられた秘密メモリの技術が開示されている。この例では秘密メモリの中のデータへアクセスするために予め定められた参照値を必要とするとしている。しかしながら、同一のＣＰＵで動作する複数のプログラム、特にＯＳから、秘密データのアクセス権を得るための参照値をどのように保護するかはこの公知技術においては開示されていない。

【００２３】

さらに、米国特許第5,123,045号（Ostorovsky et.al、以下従来技術３と呼ぶ）では、アプリケーション対応に固有の秘密鍵を持つサブプロセッサを前提とし、それらサブプロセッサがメインメモリ上に置かれたプログラムをアクセスする時にそのアクセスパターンからプログラムの動作を推定されることのないシステムが開示されている。メモリに対して操作を行う命令体系を、それとは異なる命令体系に変換し、ランダムなメモリアクセスが行われるのがその機構である。

【００２４】

しかしながら、この技術はアプリケーション毎に異なるサブプロセッサが必要となり、コスト高になるばかりでなく、このような命令体系を処理するコンパイラやプロセッサハードウェアの実装、高速化技術は、現在のプロセッサのそれとはかなり異なり、非常に困難なものとなることが予想される。なによりも、このようなプロセッサでは、実際に動作するコードの動作やデータを観察、追跡してもデータの内容や動作の対応関係の把握が著しく困難となるため、プログラムコードやデータが単純に暗号化されている前の公知技術１，２と比較して、プログラムのデバッグが非常に困難となる点で、実用には多くの課題がある。

【００２５】

そこで、これらの問題点を解決するために、本発明の第1の目的は、マルチタスク環境下において、割り込みにより実行が中断された場合でも、内部の実行アルゴリズムとメモリ領域内部のデータ状態の双方を、不正な解析から確実に保護することのできるマイクロプロセッサの提供にある。この目的は、従来技術ではプログラムコードの値を保護することはできたが、例外発生やデバッグ機能によるプログラム実行の中断を利用した解析を防ぐことはできなかったという問題に鑑みてている。そこで、プログラム実行中断時にも、確実にコードを保護することができ、かつ、その保護が、現代のＯＳが必要とする実行制御機能およびメモリ管理機能の双方と両立するマイクロプロセッサの提供を目的とする。

【００２６】

本発明の第２の目的は、異なる暗号化鍵で暗号化された複数のプログラムが実行される場合にも、それぞれのプログラムが互いに独立して、正しく読み書きできるデータ領域を確保したマイクロプロセッサの提供にある。特に、Hartmanによる米国特許第5,224,166号（従来技術１）では、暗号化されていないコードによる暗号化されたデータ領域へのアクセスが禁止されるだけの単純な保護しか行なわれておらず、複数のプログラムがそれぞれ独立に秘密を保護することができなかった。この問題点に鑑みて、複数のアプリケーションがそれぞれ秘密を持つ（暗号化されている）場合、各々の秘密をＯＳから守るデータ領域を有するマイクロプロセッサの提供を第２の目的とする。

【００２７】

本発明の第３の目的は、第２の目的で確保したデータ領域の保護属性（すなわち暗号化属性）を、ＯＳによる不正な書き換えから保護することのできるマイクロプロセッサの提供にある。従来技術１では、プログラムがメモリ上にデータを書きだす直前に、ＯＳがコンテキスト切替を利用してプログラムの実行を中断し、セグメントレジスタに設定された暗号化属性を書き換えることができるという欠点があった。暗号化属性を書き換えることによって平文でデータが書き込まれる状態になると、データは暗号化されずにメモリに書き込まれてしまう。アプリケーションがあるタイミングでセグメントレジスタ値をチェックしたとしても、その後にレジスタ値が書き換えられてしまえば同じことである。そこで、このような改変を禁止または検出して対抗措置をとることのできる機構を備えたマイクロプロセッサの提供を第３の目的とする。

【００２８】

本発明の第４の目的は、プログラムがデータ暗号化鍵として任意の値を使用することができ、暗号解析理論上のいわゆる「選択平文攻撃」から暗号化属性を守ることのできるマイクロプロセッサの提供にある。

【００２９】

本発明の第５の目的は、プログラムデバッグ、フィードバックのための機構を備えたマイクロプロセッサを提供することにある。すなわち、マイクロプロセッサ自体に平文でプログラムのデバッグを行わせ、また、実行に失敗した場合にプログラムコードの提供者（プログラムベンダ）に不具合情報をフィードバックさせることを意図する。

【００３０】

本発明の第６の目的は、上記第１～第５の目的を、安価かつ高性能に実現することのできるマイクロプロセッサの提供にある。

【００３１】

【課題を解決するための手段】

本発明の第１の特徴として、上記第１の目的を達成するために、１チップまたは１パッケージとして構成されるマイクロプロセッサは、読み出し手段としてのバスインターフェイスユニットを介して、プログラムごとに異なるコード暗号化

鍵で暗号化された複数のプログラムを、マイクロプロセッサ外部の記憶手段（たとえばメインメモリ）から読み出す。マイクロプロセッサの復号化手段は、読み出した複数のプログラムを、それぞれ対応する復号化鍵で復号化し、命令実行部が、復号化された複数のプログラムを実行する。複数のプログラムのうち、あるプログラムの実行を中断する場合に、実行状態書き込み手段としてのコンテキスト情報暗号化／複号化ユニットは、中断されるプログラムのそれまでの実行状態を示す情報と、このプログラムのコード暗号化鍵とを、マイクロプロセッサに固有の暗号化鍵で暗号化し、暗号化した情報をコンテキスト情報としてマイクロプロセッサ外部の記憶手段に書き込む。中断されたプログラムを再開する場合に、再開手段としての検証ユニットは、暗号化されたコンテキスト情報を、マイクロプロセッサに固有の暗号化鍵に対応する固有の復号化鍵で復号化し、復号化されたコンテキスト情報に含まれるコード暗号化鍵（すなわち再開予定のプログラムのコード暗号化鍵）が、中断されたプログラム本来のコード暗号化鍵と一致した場合にのみ、プログラムの実行を再開する。

【００３２】

また、第２および第３の目的を達成するために、マイクロプロセッサは、外部へ読み出すことのできないプロセッサ内部の記憶領域（たとえばレジスタ）と、プログラムの処理対象であるデータのための暗号化属性を、内部の記憶手段に書き込む暗号化属性書き込み手段（たとえば命令ＴＬＢ）とを、さらに有する。暗号化属性とは、たとえば、プログラムのコード暗号化鍵と、暗号化の対象となるアドレス範囲である。このような暗号化属性の少なくとも一部は、コンテキスト情報に含まれる。

【００３３】

コンテキスト情報暗号／複号化ユニットはまた、マイクロプロセッサに固有の秘密情報に基づく署名をコンテキスト情報に付与する。この場合、検証ユニットは、復号化されたコンテキスト情報に含まれる署名が、マイクロプロセッサに固有の秘密情報に基づく本来の署名と一致するかどうか判断し、一致する場合にのみ中断されていたプログラムを再開する。

【００３４】

このように、暗号化プログラムによるそれまでの実行状態は、コンテキスト情報として外部メモリに保存されるとともに、実行処理対象であるデータの保護属性は、プロセッサ内部のレジスタに格納されているので、データに対する不正な改変が防止される。

【００３５】

本発明の第２の特徴として、上記第４の目的を達成するために、１チップまたは１パッケージとして構成されるマイクロプロセッサは、外部へ読み出すことのできない固有の秘密鍵を内部に保持する。読み出し手段としてのバスインターフェイスユニットは、あらかじめ秘密鍵に対応したマイクロプロセッサに固有の公開鍵によって暗号化されたコード暗号化鍵を、マイクロプロセッサ外部の記憶手段から読み出す。第１の復号化手段としての鍵復号化ユニットは、読み出したコード暗号化鍵を、マイクロプロセッサの秘密鍵を用いて復号化する。バスインターフェイスユニットはまた、それぞれが異なるコード暗号化鍵で暗号化された複数のプログラムを、外部の記憶手段から読み出す。第２の復号化手段としてのコード復号化ユニットは、読み出された複数のプログラムを復号化する。命令実行部は、復号化された複数のプログラムを実行する。複数のプログラムのうち、あるプログラムの実行が中断される場合に、乱数発生機構は、一時的な鍵として乱数を発生する。コンテキスト情報暗号／複号化ユニットは、中断されるプログラムの実行状態を示す情報を生成された乱数で暗号化した第１の値と、この乱数を中断するプログラムのコード暗号化鍵で暗号化した第２の値と、乱数をマイクロプロセッサの秘密鍵で暗号化した第３の値とを、コンテキスト情報として外部の記憶手段に書き込む。

【００３６】

プログラムの実行を再開するときは、コンテキスト情報暗号／複号化ユニットは、外部の記憶手段から前記コンテキスト情報を読み出し、秘密鍵を用いて、コンテキスト情報に含まれる第３の値の乱数を復号化し、復号化された乱数を用いて、コンテキスト情報に含まれる実行状態情報を復号化する。同時に、再開される予定のプログラムのコード暗号化鍵を用いて、コンテキスト情報に含まれる第２の値の乱数を復号化する。第２の値をコード暗号化鍵で復号化した乱数と、第

３の値を秘密鍵で復号化した一時鍵とを比較し、一致する場合にのみ、プログラムの実行を再開する。

【００３７】

このように、それまで実行されていた状態を示すコンテキスト情報は、保存の都度生成される乱数で暗号化され、マイクロプロセッサに固有の秘密鍵による署名が添付されるので、安全なかたちで外部メモリに保存される。

【００３８】

本発明の第３の特徴として、前記第１～第３および第６の目的を達成するために、１チップまたは１パッケージとして構成されるマイクロプロセッサは、プログラムごとに異なる前記暗号化鍵によって暗号化された複数のプログラムを、マイクロプロセッサ外部の記憶手段から読み出して実行する。このマイクロプロセッサは、外部へ読み出すことのできない内部記憶手段（たとえばレジスタ）を有し、レジスタに、各プログラムから参照される（すなわち処理対象となる）データのための暗号化属性と暗号化属性特定情報とを格納する。コンテキスト情報暗号／複号化ユニットは、レジスタに格納された暗号化属性特定情報と関連し、マイクロプロセッサに固有の署名を含む関連情報を、外部の記憶手段に書き込む。保護テーブル管理部は、プログラムが参照するデータのアドレスに基づいて、外部の記憶手段から関連情報を読み込む。検証ユニットは、読み込まれた関連情報に含まれる署名を、秘密鍵によって検証し、それがマイクロプロセッサ固有の署名と一致した場合にのみ、暗号化属性特定情報と読み込まれた関連情報に基づいて、プログラムによるデータの参照を許可する。

【００３９】

この構成では、内部レジスタに格納されるべき情報に署名をつけて、外部メモリに書き込んでおき、必要な部分だけをマイクロプロセッサに読み込む。読み込み時に、署名を検証するので、すり替えなどに対する安全性が確保される。扱うプログラムの数が増え、暗号化属性の種類が増大した場合にも、マイクロプロセッサ内部の記憶領域を拡張する必要がなく、コストを低減できる。

【００４０】

本発明の、その他の特徴、効果は、図面を参照して以下で述べる詳細な説明に

より、明らかになるものである。

【００４１】

【発明の実施の形態】

（第１実施形態）

図１～１４を参照して、本発明の第１実施形態にかかるマイクロプロセッサについて説明する。本実施形態では、広く普及しているIntel社のPentium Proマイクロプロセッサ（インテル・アーキテクチャ・ソフトウェア・ディベロッパーズ・マニュアル参照）に変更を加えたアーキテクチャを例にとって本発明の構成を説明するが、これは本発明の適用範囲を前記プロセッサに制限するものではない。なお、Intel Pentium Proプロセッサに特徴的な部分や、他のアーキテクチャへの適用については、可能な限り本文中で補足する。

【００４２】

また、Pentium Proアーキテクチャではアドレス空間に物理アドレス、リニアアドレス、論理アドレスの３種類の区分があるが、本実施例ではPentiumの用語におけるリニアアドレスを、論理アドレスという呼び方で使用するものとする。

【００４３】

以下の説明では、特に断らない限り、保護とはアプリケーションの秘密の保護（すなわち暗号化による保護）を意味する。したがって、通常使われる保護の概念、すなわちあるアプリケーションの動作によって他のアプリケーションの動作が妨害されないこととは区別される。ただし、本発明においては、アプリケーションの秘密の保護と並列して、通常の意味での動作保護機構がオペレーションシステム（ＯＳ）によって当然に提供されているものとする（ただし、後者に関しては本発明と直接の関係がないので、説明は省略する。）。

【００４４】

また、以下の実施形態では、秘密保護をＯＳの管理下のアプリケーションの秘密を守るものとして説明しているが、この機構はＯＳそのものを改竄や解析から守る機構としても利用可能である。

【００４５】

図１は本発明の第１実施形態にかかるマイクロプロセッサの基本構成図であり

、図２は、図１に示したマイクロプロセッサの詳細構成図である。

【００４６】

マイクロプロセッサ１０１は、プロセッサコア１１１、命令ＴＬＢ（変換索引バッファ）１２１、例外処理部１３１、データＴＬＢ１４１、２次キャッシュ１５２を含む。プロセッサコア１１１は、バスインタフェースユニット１１２、コード・データ暗号化／復号化処理部１１３、１次キャッシュ１１４、および命令実行部１１５を有する。命令実行部１１５は、命令フェッチ／デコードユニット２１４と、命令プール２１５と、命令実行切替ユニット２１６と、命令実行完了ユニット２１７を有する。

【００４７】

例外処理部１３１は、レジスタファイル２５３、コンテキスト情報暗号／福号化ユニット２５４、例外処理ユニット２５５、秘密保護違反検出ユニット２５６、コード暗号化鍵・署名検証ユニット２５７を含む。命令ＴＬＢ１２１は、ページテーブルバッファ２３０と、コード復号化鍵バッファ２３１と、鍵復号化ユニット２３２を有する。データＴＬＢ１４１は、保護テーブル管理部２３３を有する。

【００４８】

マイクロプロセッサ１０１は、このマイクロプロセッサに固有の公開鍵Ｋｐと秘密鍵Ｋｓを保存する鍵記憶領域２４１を有する。今、あるプログラムベンダから所望の実行プログラムＡを購入し、実行する場合を考える。プログラムベンダは、実行プログラムＡを供給する前に、コード暗号化の共通鍵Ｋcodeでプログラムを暗号化し（$E_{Kcode}[A]$）、暗号化に用いた共通鍵Ｋcodeをマイクロプロセッサ１０１の公開鍵Ｋｐで暗号化して（$E_{Kp}[Kcode]$）マイクロプロセッサ１０１に送ってくる。マイクロプロセッサは、この実行プログラムＡだけではなく、複数の異なる暗号化プログラムを擬似並列的に（すなわち割り込み中断可能に）処理するマルチタスクプロセッサである。また、暗号化プログラムだけではなく、平文プログラムの実行も当然行う。

【００４９】

マイクロプロセッサ１０１は、バスインターフェイスユニット（読み出し手段

）１１２を介して、プログラムごとに異なるコード暗号化鍵で暗号化された複数のプログラムを、マイクロプロセッサ外部のメインメモリから読み出す。コード復号化ユニット２１２は、読み出した複数のプログラムを、それぞれ対応する復号化鍵で復号化し、命令実行部１１５が、復号化された複数のプログラムを実行する。あるプログラムの実行を中断する場合に、例外処理部１３１のコンテキスト情報暗号化／複号化ユニット２５４は、中断されるプログラムのそれまでの実行状態を示す情報と、このプログラムのコード暗号化鍵とを、マイクロプロセッサの公開鍵で暗号化し、暗号化した情報をコンテキスト情報としてメインメモリ２８１に書き込む。中断されたプログラムを再開する場合は、コード暗号化鍵・署名検証ユニット２５７は、暗号化されたコンテキスト情報を、マイクロプロセッサの秘密鍵で復号化し、復号化されたコンテキスト情報に含まれるコード暗号化鍵（すなわち再開予定のプログラムのコード暗号化鍵）が、中断されたプログラム本来のコード暗号化鍵と一致するかどうかを検証し、一致した場合にのみ、プログラムの実行を再開する。

【００５０】

ここで、マイクロプロセッサ１０１の詳細な構成、機能を説明する前に、まずマイクロプロセッサ１０１による平文の命令の実行と、暗号化プログラムの実行における処理の流れをおおまかに説明する。

【００５１】

マイクロプロセッサ１０１が平文命令を実行する時は、命令フェッチ／デコードユニット２１４が、プログラムカウンタ（不図示）の示すアドレスの内容をＬ１命令キャッシュ２１３から読み出そうとする。指定のアドレスの内容がキャッシュされていればＬ１命令キャッシュ２１３から命令を読み出し、命令プール２１５に送って命令が実行される。命令プールは複数の命令を並列実行可能であり、実行を行うためのデータの読み出しを、命令実行切替ユニット２１６に要求してデータを受け取る。命令が並行して実行され、その実行結果が確定すれば、実行結果は命令実行完了ユニット２１７に送られる。命令実行完了ユニット２１７は、操作対象がマイクロプロセッサ１０１内部のレジスタであれば、実行結果をレジスタファイル２５３に書き込み、操作対象がメモリであれば、Ｌ１データキ

ャッシュ２１８に書き込む。

【００５２】

Ｌ１データキャッシュ２１８の内容は、バスインタフェースの制御下にあるＬ２キャッシュ１５２でさらにもう一度キャッシュされて、メインメモリ２８１に書き込まれる。ここでは仮想記憶機構が使われており、論理的なメモリアドレスと物理的なメモリアドレスとの対応関係を定義するのが図３に示すページテーブルである。ページテーブルは、物理メモリ上におかれるデータ構造である。データＴＬＢ１４１は、実際に論理アドレスから物理アドレスへの変換を行うと同時に、データキャッシュを管理する。データＴＬＢ１４１は、マイクロプロセッサ１０１内部のあるレジスタが示すテーブルの先頭アドレスに基づいて、テーブルの必要な部分を読み込み、論理アドレスから物理アドレスへの変換作業を行う。このとき、メモリ上のページテーブルの全てがデータＴＬＢ１４１に読み込まれるのではなく、アクセスされる論理アドレスに応じて必要な部分だけがページテーブルバッファ２３４に読み込まれる。

【００５３】

キャッシュ動作の基本は、プログラムの命令が暗号化されているかどうかにかかわらず、一定である。すなわち、命令ＴＬＢ１２１にページテーブルの一部が読み込まれ、その定義に従ってアドレス変換が行われる。バスインタフェースユニット１１２は、メインメモリ２８１またはＬ２キャッシュ１４１から命令を読み込み、Ｌ１命令キャッシュ２１３に命令が格納される。Ｌ１命令キャッシュ２１３への命令の読み出しは、複数のワードで構成されるラインと呼ばれる単位で行われ、ワード単位の読み出しよりも高速なアクセスが行われる。

【００５４】

実行命令の演算処理対象であるデータについても、物理メモリ上にある同じページテーブルを利用してアドレス変換が行われるが、変換の実行は上述したようにデータＴＬＢ１４１で行われる。

【００５５】

ここまでは一般のキャッシュメモリの動作と基本的に同一であり、プロセッサの動作とキャッシュメモリについての詳細は、Curt Schimmel著の「ＵＮＩＸカ

ーネル内部解析」（ソフトバンク社、１９９６年）に記載されている。

【００５６】

次に、暗号化されたプログラムを実行する場合の動作を説明する。本発明では秘密の保護を受ける実行コードは全て暗号化されていることを前提とし、暗号化されている実行コードのことを保護されたコードとも言う。さらに、同一の暗号化鍵による保護の範囲を保護ドメインと呼ぶ。すなわち、同じ鍵で保護されるコードの集合は同じ保護ドメインに所属し、異なる暗号化鍵で保護されるコードは保護ドメインが異なると言う。

【００５７】

まず、メインメモリ２８１上には、共通鍵方式のブロック暗号アルゴリズムによって暗号化されたプログラムの実行コードが格納されている。プログラムベンダから送られてきた暗号化プログラムのロード方法などについては、同一出願人による特許出願第２０００－３５８９８号に記載されるとおりである。暗号ブロックサイズは、ブロックサイズの２のべき乗倍が、キャッシュメモリの読み書きの単位であるラインサイズと一致すれば、どのような値をとってもよい。ただし、ブロックサイズが小さい場合、ブロック長と命令長が一致して、サブルーチンの先頭部分などの予測可能な命令部分と暗号化データの対応関係を記録することによって、簡単に命令が解読されてしまう危険がある。そこで、本発明ではブロックをインタリーブして、ブロック中のデータの間に依存性を持たせ、暗号化されたブロックに複数の命令語やオペランドの情報を含ませる。こうすることにより、命令と暗号化ブロックの対応づけを困難にする。

【００５８】

図５は、本発明で行うインタリーブの例を示す。図５に示す例では、キャッシュのラインサイズを３２バイト、ブロックサイズを６４ビット（すなわち８バイト）としている。インタリーブ前は１ワードは４バイトで構成され、ワードＡはＡ０～Ａ３の４バイトからなる。１ラインはＡ～Ｈの８ワードで構成されている。これをブロックサイズ６４ビットに対応する８バイト単位になるようにインタリーブすると、図５の下の図のように、Ａ０，Ｂ０，…，Ｈ０がワード０および１に該当する最初のブロックに配置され、Ａ１，Ｂ１，…，Ｈ１が次のブロック

に配置される。

【００５９】

インタリーブを施す領域の長さは、長ければ長いほど攻撃は困難となるが、ラインサイズを越えてインタリーブをかけることは、あるキャッシュラインの復号化／暗号化が別のラインの読み出し／書き込みに依存することになり、処理の複雑化と速度低下を招くことになる。インタリーブを施す範囲はキャッシュラインサイズの範囲内にとどめることが望ましい。

【００６０】

ここではキャッシュラインに含まれる複数のブロックのデータに依存性を持たせるためにブロックのデータをインタリーブする方法をとっているが、データブロック間に依存性を持たせるための他の方法、例えばブロック暗号のＣＢＣ（Cipher Block Chaining）モードなどを使ってもよい。

【００６１】

暗号化された実行コードの復号化鍵Ｋcode（共通鍵アルゴリズムでは暗号化鍵と復号化鍵は同一なので、以下では復号に用いる場合も暗号化鍵という言葉を使う）は、ページテーブルに基づいて決定される。図３および４は、論理アドレスから物理アドレスへの変換テーブル構造を示す。

【００６２】

プログラムカウンタの論理アドレス３０１がある値を示し、その上位ビットのディレクトリ３０２とテーブル３０３によって、ページエントリ３０７－ｊが指定される。ページエントリ３０７－ｊは、キーエントリＩＤ３０７－ｊ－Ｋを含み、このＩＤに基づいて、キーテーブル３０９内で、このページの復号化に使用される鍵エントリ３０９－ｍが決定される。キーテーブル３０９の物理アドレスは、マイクロプロセッサ内部のキーテーブル制御レジスタ３０８によって指定される。

【００６３】

この構成では、ページエントリに直接鍵情報をおかずに、鍵エントリのＩＤを置くことにより、サイズの大きい鍵情報を複数のページで共有して、サイズの限られる命令ＴＬＢ１２１上のメモリ領域を節約することができる。

【００６４】

本発明の特徴として、キーテーブルのエントリは固定長だが、それぞれのテーブルで使用される鍵の長さは暗号解析能力の向上に対応できるように可変長とし、キーテーブルのキーサイズ領域で指定される。マイクロプロセッサ１０１に固有の秘密鍵Ｋｓは固定されているが、プログラムの暗号化、復号化に用いられるＫcodeの長さは鍵エントリの指定によって変えられるということである。可変長の鍵の位置を指定するため、鍵エントリ３０９－ｍには鍵エントリをポイントするフィールド３０９－ｍ－４があり、キーオブジェクト３１０のアドレスを示している。

【００６５】

キーオブジェクト領域３１０には、実行コードの暗号化鍵Ｋcodeが、公開鍵アルゴリズムにより、マイクロプロセッサ１０１の公開鍵Ｋｐで暗号化された形E_{Kp}[Kcode]で格納されている。公開鍵アルゴリズムでデータを安全に暗号化するためには、大きな冗長度が必要なため、暗号化されたデータ長は元のデータ長より長くなる。ここでは、Ｋｓ、Ｋｐの長さを１０２４ｂビット、Ｋcodeの長さを６４ｂビットとし、パディングにより２５６ビットの長さとして、Ｅ[Kcode]を１０２４ビットの長さに暗号化してキーオブジェクト領域３１０に格納する。Ｋcodeが長く、１０２４ビットに格納できない場合は、複数の１０２４ビットブロックに分割して格納する。

【００６６】

図６は、上記の流れをまとめたものである。プログラムカウンタ５０１は、論理アドレス空間５０２の上の暗号化されたコード領域５０３の上のアドレスAddrを指している。論理アドレスAddrは、命令ＴＬＢ１２１に読み込まれたページテーブル３０７に基づいて、物理アドレスAddr’に変換される。これと同時に、暗号化されたコード復号化鍵Ｅ[Kcode]がキーテーブル３０９から取り出され、復号化機能５０６においてＣＰＵの持つ秘密鍵Ｋｓで復号化されて、カレントコード復号化鍵記憶手段５０７に格納される。コード暗号のための共通鍵Ｋcodeは、プログラムベンダにより、マイクロプロセッサ１０１の公開鍵Ｋｐで暗号化され、Ｋcodeで暗号化されたプログラムとともに供給されるので、マイクロプロセ

ッサ１０１の秘密鍵Ｋｓを知らないユーザは、Ｋcodeを知ることはできない。

【００６７】

プログラムベンダは、Ｋcodeでプログラムの実行コードを暗号化し出荷した後は、Ｋcodeを安全に保管し、第３者に秘密が洩れないように管理する。

【００６８】

全体キーテーブル５１１と、全体ページテーブル５１２は、物理メモリ５１０に置かれ、それぞれの物理アドレスがキーテーブルレジスタ５０８、ＣＲ３レジスタ５０９によって指定されている。これらの全体テーブルの内容は、バスインタフェースユニット１１２を介して、必要な部分だけが命令ＴＬＢ１２１にキャッシュされる。

【００６９】

さて、命令ＴＬＢ１１２によって変換された物理アドレスAddr’に対応する内容５０３がバスインタフェースユニット１１２によって読み出されると、このページは暗号化されているので、コード復号化ユニット２１２で復号化される。読み出しは前記キャッシュラインサイズ単位で行われ、ブロック単位で復号化された後、上述したインタリーブの逆処理が行なわれる。復号された結果はＬ１命令キャッシュ２１３に格納され、命令として実行される。

【００７０】

その他、プログラムのロード手法、リロケーションなどについては、同一出願人による特許出願第２０００－３５８９８号に記載されている。

【００７１】

このように暗号化されたコードを実行できることにより、本発明のマイクロプロセッサでは、コードを逆アセンブルしてプログラムの動作を解析することを、暗号化アルゴリズムとパラメータを適切に選びさえすれば暗号学的に不可能とすることができる。

【００７２】

同様に、ユーザはコード暗号化鍵Ｋcodeの真の値を知ることができないので、暗号化されたプログラムの一部を改変してユーザがそのアプリケーションが扱うコンテンツの不正コピーを取るなどのユーザの意図に沿った改変も暗号学的に不

可能にすることができる。

【００７３】

次に、デバッグ機能の抑止について説明する。

【００７４】

命令ＴＬＢ１２１は、現在実行中のコードが保護されているかどうか（暗号化されているかどうか）を判定することができる。保護されたコードの実行中は、デバッグフラグやデバッグレジスタから暗号化プログラムの解析に侵入されることを防止するため、デバッグレジスタ機能と、ステップ実行機能という２つのデバッグ機能が禁止される。

【００７５】

デバッグレジスタ機能とは、プロセッサに備えられたデバッグレジスタに、メモリアクセス範囲や、実行、データとしての読み出し、書き込みなどのアクセス種別を設定しておくことにより、対応するメモリアクセスが発生したときに割り込みが発生する機能である。本実施形態においては、保護されたコードを実行している時は、デバッグレジスタに設定された内容は無視され、デバッグのために割り込みは発生しない。ただし、ページテーブルにデバッグのビットが設定されている場合を除く。ページテーブルのデバッグビットについては後述する。

【００７６】

非保護（平文）のコードを実行中は、プロセッサのＥＦＬＡＧＳレジスタのステップ実行ビットをセットすると、１命令を実行する度に割り込みが発生するが、保護されたコードの実行中には、このビットも無視され、割り込みは発生しない。

【００７７】

本発明では、実行コードの暗号化による解析の防止に加えて、これらの機能により、デバッグレジスタやデバッグフラグによるプログラムの動的解析を防止することで、ユーザによるプログラムの解析を困難にしている。

【００７８】

次に、本発明のマイクロプロセッサの特徴である、マルチタスク環境下でのプログラム実行中断時における、コンテキストの暗号化と署名および検証について

説明する。

【００７９】

マルチタスク環境でプログラムの実行は、例外によりしばしば中断される。通常、実行が中断された時にはプロセッサが保持する状態は一旦メモリ上に保存され、後でそのプログラムの実行が再開される時に元の状態が復帰される。これにより、複数のプログラムによる処理を疑似的に並行して実行したり、割り込み処理を受け付けることが可能になっている。この中断時の状態情報は、コンテキスト情報と呼ばれる。コンテキスト情報にはアプリケーションが使用するレジスタ情報が含まれ、それに加えて、明示的には見えないがレジスタの情報が含まれる場合がある。

【００８０】

従来のプロセッサでは、あるプログラムの実行中に割り込みが発生すると、アプリケーションのレジスタ状態が保持されたままＯＳの実行コードに制御が移されるので、ＯＳはそのプログラムのレジスタ状態を調べてどのような命令を実行していたかを推定したり、平文のまま保存されたコンテキスト情報を実行の中断中に改変することで、そのプログラムの実行再開後のプログラムの動作を変えることができた。

【００８１】

そこで、本発明では、保護されたコードの実行中に割り込みが発生した時に、その直前に実行していたコンテキストを暗号化して保存し、全てのアプリケーションレジスタを暗号化または初期化するとともに、コンテキスト情報にプロセッサによる署名を添付する。中断からの復帰時に署名を検証し、その署名が正しい署名かどうかをチェックする。署名の不整合を検出した場合には復帰を中止して、ユーザによる不正なコンテキスト情報の改変を防止することができる。このとき、暗号化の対象となるレジスタは、図７の７０１～７２０までのユーザレジスタである。

【００８２】

Pentium Proアーキテクチャでは、プロセスのコンテキスト情報のメモリ上への保存、復帰をハードウェア的に支援する機構がある。この状態を保存するため

の領域はＴＳＳ（task state segment）と呼ばれる。以下この機構に本発明を適用した例を説明するが、これは本発明の適用をPentium Proアーキテクチャに限定するものではなく、本発明は広く一般のプロセッサアーキテクチャに適用が可能である。

【００８３】

例外発生にともなうコンテキスト情報の保存は次の場合に起きる。例外が発生すると、ＩＤＴ（Interrupt Descriptive Table）と呼ばれる例外処理を記述するテーブルの中から、割り込み原因に対応するエントリが読み出され、そこに記述された処理が実行される。エントリがＴＳＳを示すものであるとき、指示されたＴＳＳに保存されていたコンテキスト情報がプロセッサに復帰される。一方、それまで実行されていたプロセスのコンテキスト情報は、そのときのタスクレジスタ７２５で指定されたＴＳＳ領域に保存される。

【００８４】

この自動的なコンテキスト保存機構を使えば、プログラムカウンタやスタックポインタを含めて全てのアプリケーション状態を保存し、復帰時に署名を検証することで改竄の有無を検出することができる。しかし、自動的なコンテキストの保存を行う場合、コンテキストの切替のために大きなオーバヘッドが生じる他に、ＴＳＳを使わない割り込み処理ができないなどの問題が生じる。

【００８５】

割り込み処理のオーバヘッドを小さくする、あるいは既存プログラムとの互換性を保つためには自動的なコンテキスト保存の機構を使わないことが望ましいが、この場合には、プログラムカウンタはスタック上に保存されてしまい、検証の対象とはできないため、悪意のあるＯＳによる改竄の対象ともなりうる。両者は目的に応じて使い分けられることが好ましい。そこで、本発明のマイクロプロセッサでは、保護された（暗号化された）実行コードに対しては、安全性を重視して、自動的なコンテキスト保存を採用する。自動的に保存するレジスタは必ずしも全てのレジスタでなくてもよい。

【００８６】

本実施例におけるコンテキスト保存と再開処理の特徴は次の３点である。

【００８７】

（１）コンテキストを生成したマイクロプロセッサ自体と、コンテキストを生成したプログラムの暗号化鍵Ｋcodeを知る者だけが、保存されたコンテキストの内容を復号化できる。

【００８８】

（２）あるコード暗号鍵Ｘで保護されたプログラムが中断され、そのコンテキストが保存された場合、その再開処理は非保護のプログラムや別のコード暗号化鍵Ｙで暗号化されたプログラムの再開に適用することはできない。すなわち、中断から復帰させるべきプログラムが、再開時に別のプログラムに入れ代えられることはない。

【００８９】

（３）改変されたコンテキストの復帰を禁止する。すなわち、仮に保存されていたコンテキストが改変されていた場合、そのコンテキストは復帰されない。

【００９０】

上記の特徴（１）により、コンテキスト情報の安全性を維持すると同時に、プログラムベンダによるコンテキスト情報の解析は可能にしておくことができる。プログラムベンダがコンテキスト情報を解析する権利を保持していることは、ユーザの使用条件で発生した不具合の原因を解析し、プログラムの品質を維持するためにも重要だからである。

【００９１】

特徴（２）は、たとえば攻撃者が、プログラムＡの実行によって生成されたコンテキストを、別の暗号プログラムＢに適用して、コンテキストに保存された既知の状態からプログラムを再開することによって、プログラムＢに含まれるデータやコードの秘密を解析したり動作を改変することを防ぐためのものである。この機能はまた、後述するデータ保護に際して、複数のアプリケーションが互いに独立して、それぞれ排他的に暗号化データを保持するための前提となる。

【００９２】

特徴（３）により、プログラムの再開時を利用したコンテキスト情報の改変を厳密に排除することができる。

【００９３】

このような機能を持たせる理由は、単にプロセッサの秘密情報に基づいてコンテキスト情報を暗号化しただけでは、秘密情報を、攻撃者の意図に沿ったコンテキスト情報の改変から守ることはできても、コンテキストが無秩序に改変され、ランダムに誤りが生じた状態からプログラムが再開される可能性を排除することができないからである。

【００９４】

以下、上記３つの特徴をそなえたコンテキスト保存と検証方法について詳細に説明する。

【００９５】

＜コンテキスト保存処理＞

図８は、第１実施形態のコンテキスト保存形式を概念的に示した図である。保護されたプログラムの実行中にハードウェア、またはソフトウェアが原因の割り込みが発生したとする。割り込みに対応するＩＤＴエントリがＴＳＳを示すものであれば、それまでのプログラムの実行処理が暗号化され、コンテキスト情報として（そのＴＳＳではなく）カレントのタスクレジスタ７２５が示すＴＳＳに保存される。そして、ＩＤＴエントリが示すＴＳＳに保存された実行状態がプロセッサに復帰される。ＩＤＴのエントリがＴＳＳを示していない場合、カレントのレジスタの暗号化または初期化のみが行なわれ、ＴＳＳの保存は行なわれない。当然そのプログラムの再開は不可能となる。ただし、ＯＳの動作の継続のため、レジスタの暗号化または初期化の対象から、フラグレジスタの一部タスクレジスタを含むシステムレジスタは除外する。

【００９６】

図８に示すコンテキストは、実際には、その内容がインタリーブされ、ブロック毎に暗号化されてメモリ上に保存される。まず保存される情報の項目について説明する。先頭には各特権モードに対応したスタックポインタとユーザレジスタ８０２～８２５があり、その次に、ＴＳＳのサイズと暗号化の有無を示す１ワード８２６が置かれる。これはプロセッサが保存されたＴＳＳが暗号化されたものであるかどうかを示すもので、ＴＳＳが暗号化された場合でも、この領域は暗号

化されずに、平文のまま保存される。続いてデータの保護のために追加されたデータ暗号化制御レジスタ（ＣＹ０～ＣＹ３）の領域８２７～８３０と、サイズをブロック長に合わせるためのパディング８３１が置かれる。最後にコンテキストを暗号化した鍵Ｋｒを実行コードの暗号化Ｋcodeで共通鍵アルゴリズムにより暗号化した値$E_{Kcode}[Kr]$８３２、コンテキストを暗号化した鍵Ｋｒをプロセッサの公開鍵Ｋｐで暗号化した値$E_{KP}[Kr]$８３３、そして、これら全体に対するプロセッサの秘密鍵Ｋｓによる署名$S_{KS}[message]$８３４が置かれる。また、タスク間の呼出し関係を保持する前回のタスクとのリンク領域８０１は、ＯＳによるタスクスケジューリングを可能にするため平文のまま保存される。

【００９７】

これら実行コードの暗号化や署名生成は、図２に示す例外処理部１３１の中のコンテキスト情報暗号化／復号化ユニット２５４によって行なわれ、実行コードの処理対象であるデータの暗号化とは独立した機能に基づく。コンテキスト情報がＴＳＳに保存される際には、別途のデータ暗号化機能によりＴＳＳのアドレスになんらかの暗号化が指定されていたとしても、その指定は無視され、コンテキストの暗号化が行なわれたそのままの状態で保存される。データ暗号化機能の暗号化属性は、それぞれの保護された（暗号化された）プログラムに固有のものであるため、あるプログラムの再開をその機能に依存することができないためである。

【００９８】

コンテキストの暗号化にあたっては、まず平文のまま記録されるＴＳＳサイズ領域８２６のワードが、値０に置き換えられる。そして、図５と関連して説明したのと同様のインタリーブが施されて、コンテキストが暗号化される。このとき、パディング８３１は暗号化ブロックサイズに合わせてインタリーブが適切に行えるサイズに設定する。

【００９９】

ここで、レジスタ値をプロセッサの公開鍵Ｋｐまたはコードの暗号化鍵Ｋcodeで直接暗号化しないのは、プログラムベンダとプロセッサの両者による暗号化されたコンテキストの解読を可能にし、同時にユーザによるコンテキストの復号化

を禁止するためである。

【０１００】

プログラムベンダはコードの暗号化鍵Ｋcodeを知っているので、Ｋcodeを使用してE_{Kcode}[Ｋｒ]８３２を復号化して、コンテキストの暗号化鍵Ｋｒを取り出すことができる。また、マイクロプロセッサ１０１は、内部に持つ秘密鍵ＫｓでE_{KP}[Ｋｒ]８３３を復号化して、Ｋｒを取り出すことができる。すなわち、プログラムベンダは、ユーザのマイクロプロセッサの秘密鍵を知ることなく、コンテキスト情報を復号化して不具合の解析を行うことができ、マイクロプロセッサ１０１自体は、内部に持つ秘密鍵でコンテキスト情報を復号化して実行を再開することができる。いずれの鍵も持たないユーザは、保存されたコンテキスト情報を復号化することはできない。また、コンテキスト情報と、上記E_{Kcode}[Ｋｒ]、E_{KP}[Ｋｒ]に対する署名S_{KS}[message]を、マイクロプロセッサ１０１の秘密鍵Ｋｓを知らないユーザが偽造することもできない。

【０１０１】

プログラムベンダとマイクロプロセッサによる互いに独立したコンテキスト情報の復号化を可能とするために、直接Ｋcodeでコンテキスト情報を暗号化する方法も考えられる。しかし、レジスタの状態が既知の場合、コードの暗号化鍵Ｋcodeに対して既知平文攻撃が行なわれるおそれがある。すなわち、データを暗号化するための鍵の値が固定されていると、次のような問題がある。ユーザのデータ入力を読み込んで、それを一時的に作業用メモリに暗号化して書き込むというプログラムを実行していたとする。暗号化されて書き込まれるデータは、メモリ上を観察していればわかるので、ユーザは入力の値を変えて何度も入力を繰り返し、それに対応する暗号化されたデータを入手することができる。これは、暗号解析の理論における「選択平文攻撃」が可能であることを意味する。

【０１０２】

共通鍵暗号アルゴリズムにとって既知平文攻撃は致命的ではないとはいえ、これを回避するに越したことはない。そこで、例外処理部１３１の乱数発生ユニット２５２により、コンテキストの保存の都度、乱数Ｋｒを生成し、コンテキスト情報暗号／復号化ユニット２５４に供給する。コンテキスト情報暗号／復号化ユ

ニット２５４は、乱数Ｋｒを用いて、共通鍵アルゴリズムによりコンテキストを暗号化する。そして、乱数Ｋｒをコード暗号化鍵Ｋcodeで同じく共通鍵アルゴリズムにより暗号化した値Ｅ［Ｋｒ］８３２として添付する。プロセッサの公開鍵ＫｐによるＫｒの暗号化E_{Kp}［Ｋｒ］８３３は、公開鍵アルゴリズムによる。

【０１０３】

ここで、乱数は、乱数発生機構２５２によって生成される。暗号化されているのがプログラムの場合には、通常はプログラムコードには変化がなく、動作の解析をしない限り、対応する平文のコードが不正に入手されることはない。この場合、暗号を破るには「暗号文単独攻撃」を行う必要があり、暗号化鍵の探索は非常に困難である。ところが、ユーザによって入力されたデータを暗号化してメモリに格納する場合には、ユーザが入力データを自由に選択できる。このため、暗号化鍵に対して「選択平文攻撃」という、「暗号文単独攻撃」と比べてはるかに効率的な攻撃が可能となってしまう。選択平文攻撃に対しては、保護されるべき平文に「salt（塩）」と呼ばれる乱数を追加して探索空間を大きくするという対策が可能である。しかし、すべてのデータにsaltの乱数値を組み込んだ形でメモリに保存することを、アプリケーションプログラミングのレベルで実装するのは非常に繁雑であり、プログラミング効率と性能の低下を招くことになる。

【０１０４】

そこで、乱数発生機構２５３は、コンテキストの保存の都度、それを暗号化するための乱数（暗号化鍵）を発生する。暗号化鍵を任意に選択できることにより、プロセス間、プロセスーデバイス間の安全な通信を高速化できるという効果も生じる。メモリアクセス時にハードウェアによってデータを暗号化する速度は、ソフトウェアによって暗号化する速度に比べて一般にはるかに速いためである。これに反し、データ領域の暗号化鍵の値が、予め決められた値、例えば実行コードの暗号化鍵と同一のものに制限されていると、別の暗号化鍵で暗号化された他のプログラムや、デバイスとの暗号化されたデータの共有にプロセッサのデータ暗号化機能を使うことができず、プロセッサに設けられたハードウェア暗号化機能の高速性を生かすことができないからである。

【０１０５】

なお、署名８３４の生成および再開時に行われる暗号化された乱数Ｅ[Ｋｒ]８３２の復号化は、マイクロプロセッサ１０１だけが行えるという条件を満たしていれば、どのようなアルゴリズムと秘密情報に基づいてもよい。上の例では、ともにマイクロプロセッサ１０１に固有の秘密鍵Ｋｓ（これはコード暗号化鍵Ｋcodeの復号にも用いられる）を使っているが、それぞれ別の値を使ってもかまわない。

【０１０６】

また、保存されたコンテキストには暗号化の有無を示すフラグがあり、暗号化されたコンテキスト情報と暗号化されないコンテキスト情報が用途に応じて共存できる。ＴＳＳのサイズと暗号化の有無を示すフラグは平文で格納されているので、過去のプログラムとの互換性を維持するのが容易である。

【０１０７】

＜中断されたプログラムの再開処理＞

コンテキストを復帰させてプロセスを再開させる時、ＯＳは保存されたＴＳＳを示すＴＳＳデスクリプタへのジャンプまたはcall命令を発行する。

【０１０８】

図２のブロック図に戻ると、例外処理部１３１のコード暗号化鍵・署名検証ユニット２５７は、最初に署名S_{Ks}[message]８３４をプロセッサの秘密鍵Ｋｐを使って検証し、検証結果を例外処理ユニット２５５に送る。検証結果が失敗である場合は、例外処理ユニット２５５はコンテキストの再開を中止し、例外を発生させる。この検証により、コンテキスト情報が確かに秘密鍵を持つ正当なマイクロプロセッサ１０１により生成されたものであって、改竄を受けていないことが確認できる。

【０１０９】

署名の検証に成功すると、コンテキスト情報暗号／復号化ユニット２５４は、コンテキスト暗号化鍵E_{Kp}[Ｋｒ]８３３を秘密鍵Ｋｓで復号化して、乱数Ｋｒを取り出す。取り出したＫｒによりコンテキストを復号化する。一方、プログラムカウンタ（ＥＩＰ）８０９に対応するコード復号化鍵Kcodeがページテーブルバッファ２３０から取り出され、コート暗号化／復号化鍵領域２５１に送られる

。コンテキスト情報暗号／復号化ユニット２５４は、コード復号化鍵Ｋcodeで$E_{Kcode}[Kr]832$を復号化し、その結果をコード暗号化鍵・署名検証ユニット２５７に送る。コード暗号化鍵・署名検証ユニット２５７は、$E_{Kcode}[Kr]832$の復号結果が、マイクロプロセッサの秘密鍵Ｋｓによる復号化結果と一致するかどうかを検証する。この検証により、このコンテキスト情報が秘密鍵Ｋcodeで暗号化されたコードの実行により生成されたものであることが確認できる。

【０１１０】

もしコンテキスト情報に対して、このコード暗号化鍵に関する検証を行わない場合には、ユーザが適当な秘密鍵Ｋａで暗号化したコードをつくり、それを実行して得たコンテキスト情報を、別の秘密鍵Ｋｂで暗号化したコードに適用する攻撃が可能となってしまう。上記の検証はこの攻撃を排除し、保護されたコードのコンテキスト情報の安全性を保証している。

【０１１１】

この目的は、コンテキスト情報に秘密のコード暗号化鍵Ｋcodeを追加することによっても達成できるが、本発明では、コンテキスト情報を暗号化する秘密の乱数Ｋｒを、プログラムベンダの選んだコード暗号鍵Ｋcodeで暗号化した値$E_{Kcode}[Kr]$を検証に使うことで、コンテキスト情報の保存に必要なメモリの量を削減し、コンテキスト切替の高速化とメモリの節約の効果を達成している。これはまた、プログラム作成者へのコンテキスト情報のフィードバックをも可能にする。

【０１１２】

さて、コード暗号化鍵・署名検証ユニット２５７により、コード暗号化鍵の検証と署名の検証に成功すると、コンテキストがレジスタファイル２５３に復帰され、プログラムカウンタの値も復帰されるので、制御がコンテキストを生成した実行中断時のアドレスに戻される。

【０１１３】

いずれかの検証に失敗して、例外処理ユニット２５５が例外を発生させた場合は、例外の発生アドレスはジャンプまたはcall命令が発行されたアドレスを示す。また、ＩＤＴテーブルの割り込み原因の領域にＴＳＳの不正を示す値が格納さ

れ、割り込み原因となるアドレスを格納するレジスタにジャンプ先のＴＳＳのアドレスが格納される。これにより、ＯＳはコンテキスト切替失敗の原因を知ることができる。なお、再開処理を高速化するため、コンテキスト情報暗号／復号化ユニット２５４により復号された実行状態のレジスタファイル２５３へ供給と、検証ユニット２５７による検証処理を並行して行ない、検証に失敗した時は以降の処理をとりやめる構成としてもよい。

【０１１４】

乱数を介したこの暗号化方式の安全性は、使用する乱数系列の予測不可能性に依存するが、予測が困難な乱数をハードウェアによって生成する方法は例えば、小野寺他による日本国特許番号第２９８０９７６号に記載されている。

【０１１５】

プログラムベンダによるコンテキスト情報の解析は、ユーザの使用条件で発生したプログラムの不具合の原因を解析し、プログラムの品質を向上させる上で重要である。本実施例ではその点を鑑み、コンテキストの安全性とプログラムベンダによるコンテキスト情報の解析可能性の両立させる方式を説明しているが、この方式により、コンテキスト保存のオーバヘッドが増えることも事実である。

【０１１６】

また、マイクロプロセッサの署名によるコンテキスト情報の検証は、不正なコンテキスト情報で保護されたコードを、任意に選んだ値と暗号化鍵の組み合わせにより実行されることを防止しているが、この追加もやはりオーバヘッドを増やしている。

【０１１７】

プログラムベンダによるコード解析の必要性や、不正なコンテキスト情報によるプログラム再開を排除する機構が必要ない場合は、コードの暗号化鍵を特定する情報を含むコンテキスト情報をプロセッサの持つ秘密鍵で直接暗号化してもよい。これだけでコンテキストの意図的な改竄は暗号学的に不可能、かつ暗号化鍵の異なるプログラムに、コンテキスト情報が適用されることを防ぐことができる。

【０１１８】

ここで、コンテキスト保存の形式について説明を追加しておく。動作との関係については後述する。

【０１１９】

図８において、８２５－１に示すＲビットはコンテキストが再開可能かどうかを示すビットである。このビットが１にセットされている場合、上記復帰手順でコンテキストに保存された状態を復帰して実行を再開できるが、値が０の場合には、再開することはできない。これは、暗号化プログラムの実行中に不正が検出されたコンテキストの再開を防ぐことで、再開可能なコンテキストを正しい状態のものに限定できるという効果を有する。

【０１２０】

８２５－２に示すＵビットはＴＳＳがユーザＴＳＳかシステムＴＳＳかを示すフラグである。このビットが０の場合は保存されたＴＳＳはシステムＴＳＳであり、このビットが１にセットされている場合は保存されたＴＳＳはユーザＴＳＳである。上の例で説明した例外エントリからの特権の変更を伴うタスク切り替えや、タスクゲートの呼出しを通して保存、復帰されるＴＳＳはシステムＴＳＳである。一方、特権の変更を伴わないユーザによるサブルーチン呼出し命令、明示的なコンテキスト保存命令の実行によって保存されるＴＳＳはユーザＴＳＳである。システムＴＳＳとユーザＴＳＳの違いは、ＴＳＳの復帰時に現在実行中のプログラムのＴＳＳ保存場所を示すタスクレジスタが更新されるかされないかにある。システムＴＳＳの復帰では、現在実行中のプログラムのタスクレジスタが、新しく復帰されるＴＳＳの前回のタスクとのリンク領域８０１に保存され、新しいＴＳＳのセグメントセレクタがタスクレジスタに読み込まれる。一方、ユーザＴＳＳの復帰では、タスクレジスタの値の更新は行われない。ユーザＴＳＳはプログラムのレジスタ状態の保存、復帰のみを目的としており、特権モードの変更は伴わない。

【０１２１】

＜データ保護＞

次に、実行コードの演算処理対象であるデータの保護について説明する。

【０１２２】

第１実施形態では、データを保護するための暗号化属性は、マイクロプロセッサ１０１の内部に設けられたＣＹ０～ＣＹ３の４つのレジスタに定義される。図７に示す領域７１７～７２０がこれに該当する。図７では、ＣＹ０～ＣＹ２については詳細を省略し、ＣＹ３のみを詳細を示している。

【０１２３】

暗号化属性の各要素を、ＣＹ３レジスタ７１７を例にとって説明する。暗号化される領域の先頭を示す論理アドレスの上位ビットが、ベースアドレス７１７－１に指定される。領域のサイズはサイズ領域７１７－４に指定される。サイズはキャッシュライン単位で指定されるため、下位のビットには無効部分がある。データの暗号化鍵は７１７－５に指定される。共通鍵アルゴリズムを使用しているので、復号化鍵にも７１７－５を使用する。暗号化鍵の値に０が指定された時、そのレジスタが示す領域は暗号化されていないことを示す。領域の指定はＣＹ０が優先され、以下ＣＹ１～ＣＹ３の順番で優先される。例えばＣＹ０とＣＹ１の指定する領域が重なった場合、その領域ではＣＹ０の属性が優先される。また、処理対象のデータではなく、実行コードとしてのメモリアクセスではページテーブルの定義が優先される。

【０１２４】

７１７－４はデバッグビットであり、デバッグ状態でのデータ操作を暗号化状態で行うか、平文状態で行うかを選択する。デバッグビットについての詳細は後述する。

【０１２５】

図１０は、実行コードの処理対象であるデータの暗号化、復号化の流れを示す。ここでは、コードが保護された状態、すなわち暗号化された状態で実行されている時のみ、データの保護が行われる。ただし、後述するデバッグ状態でコードが実行されている場合は除外する。コードが保護されている時、暗号化属性レジスタＣＹ０～３の内容は、図２に示すレジスタファイル２５３からデータＴＬＢ１４１内にあるデータ暗号化キーテーブル２３６に読み込まれている。

【０１２６】

ある命令が論理アドレスAddrにデータを書き込む時、データＴＬＢ１４１は、

論理アドレスAddrがＣＹ０～３の範囲に含まれているかどうかを、データ暗号化キーテーブル２３６（図２参照）を調べることにより判断する。判断の結果、暗号化属性が指定されていれば、データＴＬＢ１４１は、Ｌ１データキャッシュ１１４からメモリへの対応キャッシュラインのメモリ書き出しの際に、メモリ内容を指定された暗号化鍵で暗号化するように、データ暗号化ユニット２２０に指示する。

【０１２７】

読み込みの場合も同様に、対象アドレスが暗号化属性を持っていれば、対応するＬ１データキャッシュ２１８のキャッシュラインの読み込みの際に、データ復号化ユニット２１９に対して、指定された暗号化鍵で復号化するように指示する。

【０１２８】

第１実施形態では、データ暗号化のためのデータ暗号化属性をすべて、マイクロプロセッサ１０１内部のレジスタに置き、実行の中断時にレジスタの内容をコンテキスト情報として安全な形でマイクロプロセッサ外部のメモリ（たとえば図２のメインメモリ２８１）に保存することにより、データ暗号化属性を、ＯＳの特権も含めた不正な書き換えから保護している。

【０１２９】

データの暗号化、復号化は、コンテキストの暗号化と関連して先に説明したインタリーブを受けたキャッシュライン単位で行われる。このため、Ｌ１キャッシュ１１４上のデータを１ビット書き換えただけでも、メモリ上ではキャッシュライン内の他のビットが書き換えられることになる。データの読み書きの実行はキャッシュライン単位でまとめて行われるため、オーバヘッドの増大はさほど大きくはないが、暗号化されたメモリ領域に対する読み書きは、キャッシュラインサイズ以下の単位では行えないことに注意が必要である。

【０１３０】

＜エントリゲート＞

本発明では、保護されていないコードから、保護されたコードに制御が移行できるのは、次の２つの場合に限られる。

【０１３１】

（１）再開アドレスと一致するコード暗号化鍵で暗号化された（乱数を持つ）コンテキストが再開される場合；

（２）連続したコードの実行またはjump，call命令などにより、保護されないコードから、保護されたコードのエントリゲート命令（ＥＧＡＴＥ命令）に制御が移る場合。

【０１３２】

この限定は、攻撃者がコードを任意の場所から実行することによって、コード断片の情報を得ることを防ぐためのものである。（１）の手順については、コンテキストの復帰と関連してすでに説明した。すなわち、中断前に実行していたコードのコード暗号化鍵と一致するコンテキスト情報が含まれていること、およびマイクロプロセッサ１０１が与えた正しい署名が付加されていることが検証された場合に、保護されたコードの実行制御に移る。

【０１３３】

（２）の手法は、保護されないコードから保護されたコードへと制御を移す場合には、制御の最初にエントリゲート（egate）命令と呼ぶ特殊な命令を実行しなければ、保護コートの実行に移行できないとする処理である。

【０１３４】

図９は、エントリゲート命令に基づく保護ドメインの切り替え手順を示す。マイクロプロセッサ１０１は、例外処理部１３１のカレントコード暗号化鍵記憶手段２５１（図２参照）に、現在実行中のコードの暗号化鍵を保持している。まず、ステップ６０１で、命令の実行にともなってこの鍵の値が変更されたかどうかを判断する。鍵の値の変更が検出された場合（ステップ６０１でＮＯ）、ステップ６０２に進み、変更に伴って実行されている命令がエントリゲート(egate)命令であるかどうかを調べる。エントリゲート命令であるということは、それが適正な命令であり、変更されたコードに制御を移行してもかまわないことを意味する。したがって、ステップ６０２でエントリゲート命令であると判定された場合は（ステップ６０２でＹＥＳ）、その命令を実行する。

【０１３５】

ステップ６０２でエントリゲート命令でないと判断された場合は（ステップ６０２でＮＯ）、割り込まれた命令が不適正な命令であることを意味する。この場合、ステップ６０３に進み、直前に実行されていた命令が暗号化（保護）されたものかどうかを判断する。保護されていない命令であれば、そのまま例外処理を発生させることができるが、保護されている命令である場合、一応その命令の保護を守りつつ例外処理しなければならないからである。

【０１３６】

したがって、ステップ６０３で保護されていない命令であると判断された場合は（ステップ６０３でＮＯ）、そのまま例外処理を行い、保護された命令であると判断された場合には（ステップ６０４でＹＥＳ）、保護状態を守ったまま、例外処理を発生させる。

【０１３７】

このような制御移行の制限により、平文コードから、エントリゲート命令がおかれた場所以外のコードへ、直接制御を移すことが禁止される。コンテキストを復帰させるということは、すでにそのプログラムがエントリゲートを通じて一度実行された状態を復帰させることである。したがって、保護されたプログラムを実行するには、必ずエントリゲートを通らなければならないことになる。プログラム中でエントリゲートを置く場所を最低限に押さえることにより、さまざまなアドレスからプログラムを実行してプログラムの構造を推定する攻撃を防ぐ効果がある。

【０１３８】

さらに、このエントリゲートでは、データの保護属性レジスタの初期化を行なう。エントリゲートを実行すると、図７に示す保護レジスタＣＹ０～ＣＹ３（７１７～７２０）の鍵領域（ＣＹ３では領域７１７－５）に、乱数Ｋｒがロードされる。暗号化対象先頭アドレスを０、サイズをメモリの上限までに設定し、論理アドレス空間のすべてを暗号化対象に設定する。実行コードにデバッグ属性が設定されていなければ、デバッグビット（ＣＹ３では７１７－３）は非デバッグとする。

【０１３９】

つまり、暗号化コードの実行開始時点では、すべてのメモリアクセスはエントリゲート実行時に決定された乱数Ｋｒで暗号化されることになる。また、前述の通り実行コードの暗号化制御はページテーブルの定義が優先される。この乱数Ｋｒはコンテキストの暗号化に使われる乱数とは独立に生成される。

【０１４０】

この機構により、新たに実行される保護されたプログラムは、必ず全てのメモリアクセス開始時にランダムに決定された鍵によって暗号化される設定となる。

【０１４１】

もちろんこのままではメモリ領域の全体が暗号化されたままなので、メモリを通じてシステムコールのパラメータを渡したり、他のプログラムとのデータ交換をすることができない。そこで、プログラムは順次必要なメモリ領域を平文としてアクセスできるように保護属性レジスタを設定して、自身の処理環境を整え処理を進める。優先度の低いレジスタＣＹ３は最初の乱数で暗号化される設定のままとしておき、その他のレジスタに平文アクセスの設定として暗号化鍵０を設定すれば、必要以外の領域を平文としてアクセスし、暗号化して秘密にするべきデータを誤って平文の領域に書き出してしまう危険を少なくすることができる。

【０１４２】

保護属性レジスタ以外のレジスタの内容は、エントリゲートにおける初期化でも暗号化せずに、スタックやパラメータの場所を指定するためのポインタを格納しておくことができる。ただし、レジスタに不正な値を設定してエントリゲートを呼び出すことにより、プログラムの秘密が盗まれることがないよう、エントリゲートを通して実行されるプログラムの処理には注意を払う必要がある。

【０１４３】

プログラミング的には制約され、効率が悪くなるが、安全性を重視すればエントリゲートでは保護属性レジスタ以外の汎用レジスタも含めて、フラグ、プログラムカウンタ以外のすべてのレジスタを初期化する構成としてもよい。この場合でもスタックなどのパラメータはプログラムカウンタの相対アドレス、または絶対アドレスで指定したメモリ領域を通じて受け渡すことが可能である。ただし、ここでもコンテキスト保存の場合と同様、ＯＳの動作の継続のため、レジスタの

暗号化または初期化の対象から、フラグレジスタの一部タスクレジスタを含むシステムレジスタは除外する。

【０１４４】

このように、第１実施形態にかかるマイクロプロセッサ１０１では、平文状態のプログラムから、保護されたプログラムへと制御が移る際に、最初に実行する命令をエントリゲート命令に制限し、エントリゲート命令の実行によってデータ暗号化属性レジスタを含むレジスタを初期化することで、保護された実行コードの断片的な実行を防ぎ、特にデータ保護状態の不正な設定を防いでいる。

【０１４５】

次に保護されたプログラムの実行制御について説明する。始めに保護ドメイン内に閉じた呼び出しおよび分岐について説明する。保護ドメイン内の呼び出しは通常のプログラムと全く同一である。図１１に、保護ドメイン内の呼び出しおよび分岐の概念を示す。

【０１４６】

保護ドメインのコード１１０１の実行は、保護ドメイン外のスレッド１１２１が、保護ドメインのegate（エントリゲート）命令へと分岐することにより開始される。egate命令の実行によってすべてのレジスタが初期化され、その後、プログラムの実行によって順次データ保護属性が設定される。jmp xxx命令により、保護ドメイン内の分岐先xxx１１１１に制御が移り（処理１３２２）、アドレスppp１１１２にあるcall yyy命令が実行される（処理１１２３）。スタックメモリ１１０２に呼び出し元のアドレスppp１１１２がプッシュされ、呼出先yyy１１１３へと制御が移る。呼び出し先での処理が完了して、ret命令が実行されると、スタックの戻り番地ppp１１１２へと制御が移る。実行コードの暗号化鍵が同一である間は実行制御に制限はない。

【０１４７】

次に保護ドメインから非保護ドメインへの呼び出しおよび分岐について説明する。この制御の移行には、保護ドメインから非保護ドメインへのプログラム作成者の意図しない移行を避けるためと、データ保護状態を保護するために、特殊な命令の実行および以下に説明するユーザＴＳＳの操作を行う。

【０１４８】

図１２は、保護ドメインから非保護ドメインへの呼び出しおよび分岐動作の概念図である。それぞれのドメインに、保護ドメインの実行コード１２０１と、非保護ドメインの実行コード１２０２が置かれている。また、ユーザＴＳＳ領域１２０３と、非保護ドメインとのパラメータ受渡し領域１２０４が設けられている。

【０１４９】

実行は、スレッド１２２１がegate命令を実行することに始まる。保護ドメインのプログラムは、非保護ドメインのコードを呼び出す前に、予め定められたパラメータ領域１２０４にユーザＴＳＳ領域１２０３のアドレスを保存しておく。そしてecall命令を実行して非保護ドメインのコードが呼び出される。ecall命令は２つのオペランドをとる。一つは呼出先のアドレスであり、もう一つは実行状態の保存先である。ecall命令は、呼出時のレジスタ状態（正確にはプログラムカウンタはecall発行後の状態）をオペランドｕＴＳＳで指定した領域に、これまでに説明した暗号化ＴＳＳと同じ形式で保存する。以下、この領域をユーザＴＳＳと呼ぶ。

【０１５０】

ユーザＴＳＳとシステムＴＳＳとの違いは、図８に示すユーザレジスタにおいて、ＴＳＳ上の領域８２５－２にＵフラグがセットされていることである。動作の違いについては、後述する。ユーザＴＳＳのメモリへの保存においても、システムＴＳＳへのコンテキスト情報の保存と同様、ユーザが保護レジスタＣＹ０～ＣＹ３に定義したデータ保護属性は適用されない。

【０１５１】

呼出先の非保護ドメインのコードでは、ecall命令の実行によりレジスタは初期化されているので、パラメータの受渡しができない。このため、予め定められたアドレスparam１２０４からパラメータを取得し、必要な処理を行なう。非保護ドメインの中ではプログラミングに制限はない。図１２の例では、サブルーチンｑｑｑ１２１３を呼び出している（矢印で示す処理１４２５）。例えばexxからqqqの呼出しまでの間に、スタックポインタの設定やパラメータのスタックへ

の複写を行うアダプタコードをおくことにより、サブルーチンqqqが有する呼び出しセマンティクスに、保護ドメインからの呼び出しを適応させることができる。処理結果は、メモリ上のパラメータ領域１２０４を通じて、呼出元へ送られる（処理１２２６）。サブルーチンの処理が完了すると、呼出元の保護ドメインへ制御を戻すため、sret命令が発行される（１２２７）。

【０１５２】

sret命令も、オペランドを持たないret命令とは異なり、ユーザＴＳＳを指定するオペランドを一つとる。ここではパラメータ領域param１２０４に格納されたポインタを通じて、間接的にユーザＴＳＳ１２０３を復帰情報として指定している。sret命令によるユーザＴＳＳの復帰が、システムＴＳＳの復帰と大きく異なる点は、ユーザＴＳＳを復帰してもタスクレジスタは全く影響を受けない点である。ユーザＴＳＳのタスクリンクのフィールドは無視される。sret命令のオペランドに８２５－２Ｕフラグが０のシステムＴＳＳが指定された場合は、復帰は失敗する。

【０１５３】

復帰の実行の際には、すでに説明した実行状態の復号化およびコード暗号化鍵と署名の検証が行なわれ、違反が検出された場合、秘密保護違反の例外が発生する。検証に成功すると、呼出元のecall命令の次の命令から実行が再開される。このアドレスはユーザＴＳＳの中で暗号化され署名されているので、偽造することは暗号学的に不可能である。プログラムカウンタを除くすべてのレジスタ呼出前の状態に戻されてしまうので、保護ドメインのコードは、サブルーチンexxの実行結果をパラメータ領域１２０４から取得する。

【０１５４】

保護ドメインの処理が完了して制御を非保護ドメインに移す時は、ejmp命令が使われる。ejmp命令はecallとは異なり、状態の保存は行なわない。もしecall, ejmp以外のjmp, call命令によって保護ドメインから非保護ドメインへと制御が移された場合、秘密保護違反の例外が発生して、暗号化されたコンテキスト情報がシステムのＴＳＳ領域（タスクレジスタが示す領域）に保存される。なお、このときコンテキスト情報は再開不可にマークされる。なお、保護ドメイン内のア

ドレスをejmp命令の飛び先として指定しても違反とはならない。

【０１５５】

以上が保護ドメインから非保護ドメインの呼び出し手順とそれに使用される新たに追加された命令である。

【０１５６】

アプリケーションによるユーザＴＳＳの復帰の際に、特権を持つＯＳがユーザＴＳＳをすりかえる攻撃の可能がないわけではない。しかし、そこで交換可能なＴＳＳ情報は、保護ドメインのコードの暗号化鍵が正しく管理されている限りは、必ずegateを通って実行を開始され、割り込みまたはユーザによる明示的な実行状態の保存によって保存されたコンテキスト情報だけである。このコンテキスト情報の入れ換えによって、アプリケーションの秘密が洩れる可能性は極めて小さく、かつ、攻撃者にとっては、どのようなコンテキスト情報の入れ替えを行なえばアプリケーションの秘密を取得できるかを予測するのは極めて困難である。

【０１５７】

上述した保護ドメインから非保護ドメインの呼び出し手順は、呼出先で最初に実行される命令が被呼出し側のegate命令ならば、保護ドメインの間で制御を移す手順に適用することも可能である。

【０１５８】

この時、両者の間のパラメータの受渡し領域を、予め両者の間で認証鍵交換を行なうことで共有した暗号化鍵によって暗号化しておけば、保護ドメインの間の呼出を安全に行なうことができる。

【０１５９】

＜スレッド間の制御＞

次に保護されたプログラムの実行と、その同一プログラムの制御下におけるスレッドとの関係、および、そこに生ずる問題と解決手段について説明する。

【０１６０】

上記では、暗号化された（保護された）プログラム内部での実行制御自体は、通常のプログラムと全く変わらないことを説明した。このことは、ユーザプログラムのレベルで、スレッドの切替を行なうユーザスレッドについても当てはまる

ことであり、ユーザスレッドでは特にプログラムが保護されていることを意識する必要はない。問題は、カーネルレベルで実現されるスレッドが複数存在する時に、どのようにデータの保護情報を共有するかである。

【０１６１】

始めに、ユーザスレッドとカーネルスレッドについて簡単に説明する。ユーザスレッドとは、ＯＳによる介入なしに、ユーザプログラムすなわちアプリケーション内部で仮想的に複数のスレッドを並列実行することである。それぞれのスレッドは、独立したスタックとプログラムカウンタ値を持つ。一方、カーネルスレッドは、スレッドの切替をＯＳに依存するスレッド実装である。機能的な相違としては、ユーザスレッドでは、あるスレッドが自発的にスレッド切替えをするコードを呼び出さない限り、スレッド切替えは発生しないが、カーネルスレッドでは、割り当てた実行時間を超過したスレッドの実行は強制的に停止される。詳細は前述した文献「ＵＮＩＸカーネル」を参照されたい。

【０１６２】

内部機構的には、カーネルスレッドではコンテキスト情報がＯＳに処理され、スレッドの切替が行なわれる。一方、ユーザスレッドではアプリケーションがスレッド切替のコードを持ち、そこでレジスタのコンテキスト保存領域への保存が行なわれる。この動作を行なうのはアプリケーション自身なので、コードが保護されていても全く問題なくスレッド切替の動作が可能であり、かつそれぞれのスレッドはデータ保護状態も共有することができる。唯一注意すべき点は、スレッドコンテキストを保存する領域を暗号化していない場合、スレッドコンテキストをＯＳや他のアプリケーションに読みとられたり、改竄されるおそれがあることである。スレッドコンテキスト情報は暗号化された領域に保存して、他のプログラムによる読みとりや意図的な改竄を防止することが望ましい。

【０１６３】

カーネルスレッドにおいては、スレッドの切替えはＯＳによって行なわれる。例えばあるスレッド処理がタイマ割り込みによって中断されたとき、そのスレッドが割り当てられた処理時間を使い尽くしている場合には、ＯＳのスケジューリング機能が他のスレッドを実行することで、リアルタイム処理における応答性も

高められるなどの利点がある。また、複数のカーネルスレッドが動作する機能を備えないＯＳも、従来のＵＮＩＸなどのＯＳには存在するが、ほとんどすべてのＵＮＩＸ実装が備える非同期的なシグナル配送機能は、コンテキスト切替の観点から、プロセスのメインのスレッドと独立に実行されるカーネルスレッドの一種と考えてよい。カーネルスレッドのサポートはほとんどのＯＳの実装に必要な機能である。

【０１６４】

ところが、保護されたコードに、現在のカーネルスレッドの実装を適用しようとすると問題が生じる。本発明のマイクロプロセッサでは、保存されたコンテキスト情報は暗号化により保護されており、安全性の観点から、その一部を読み出す手段は備えていない。したがって、従来の方法では、別々に起動されたカーネルスレッドがデータ保護状態を共有することはできない。カーネルスレッドはそれぞれが別々のＴＳＳを持つが、コンテキスト情報を単純に別のＴＳＳ領域にコピーしたとしても、スタック領域も含めてすべて同一となってしまうため、それらは別々のスレッドとして独立に動作することはできなくなる。

【０１６５】

そこで、本発明では、本発明のプロセッサに、プログラムによる、ある特定の時点でのコンテキスト情報の保存命令と、以下の手順によってカーネルスレッド間のデータ保護情報とを共有させ、カーネルレベルのマルチスレッド機能を可能にしている。

【０１６６】

図１４は、データ保護属性の共有に使われるスレッドテーブルを示す。スレッドＩＤは、そのカーネルスレッドのタスクレジスタの値である。スレッドＩＤ０は初期化のために特殊な役割を持つ。ユーザＴＳＳは、そのスレッドが保護ドメイン外のサブルーチンを呼ぶ場合などに、状態を保存するためのユーザＴＳＳ領域へのポインタを格納する。パラメータは、ＯＳからそれぞれのスレッドへのパラメータ渡しや、逆にスレッドから保護ドメイン外のサブルーチンを呼び出す際、パラメータの渡しに使われる。作業中フラグはテーブルおよびＴＳＳ０の書き替えの排他制御に使われる。

【０１６７】

図１３は、同一プログラム下でのデータ保護属性共有の手順を示すフローチャートである。ＯＳは保護ドメインを最初に実行するメインのスレッドの実行前に、図１４に示すスレッドテーブルを初期化する。初期化はスレッドＩＤ０のユーザＴＳＳフィールドに値０を書き込み、メインのスレッドＴＳＳ１のパラメータフィールドに、メインのスレッドの実行に必要なパラメータ、例えば引数などが格納されたパラメータブロックへのポインタを書き込むことによって行われる。

【０１６８】

ステップ１３０１で、カーネルスレッドは最初にegateを実行して保護ドメインのコードの実行を開始する。このとき、システムレジスタ以外のレジスタはすべて初期化される。次に、ステップ１３０２で、コード中に予め埋め込まれたスレッドテーブルのアドレスからスレッドテーブルを読みとり、初期化用のスレッドＩＤ０に対応するユーザＴＳＳフィールドの値を見ることによって、そのユーザＴＳＳフィールドで初期化が開始されているかどうかを判断する。ユーザＴＳＳフィールドの値が０であれば、まだ初期化されていない状態である。この場合、スレッドテーブルが未初期化であると判断して（ステップ１３０２でＮＯ）、ステップ１３０３に進む。

【０１６９】

ステップ１３０３で、スレッドＩＤ０の作業中フラグのフィールドに１を書き込み、初期化が開始されたことを示してから、データ暗号化属性を初期化する。データ暗号化属性の初期化は、実行される保護されたアプリケーションに依存する。ここではプログラムのロード時に確保されている固定アドレスの領域に、予めプログラムで決められた暗号化鍵を設定してもよいし、動的にメモリを確保して乱数による暗号化鍵を設定してもよい。データ暗号化属性の設定が完了すると、ステップ１３０４に進み、たとえばスレッド１（タスクレジスタがシステムＴＳＳ１を示している）の実行状態を、ユーザＴＳＳを保存する命令であるstctx命令により、ｕＴＳＳ０の領域に保存する。stctx命令で保存されるＴＳＳは、図８に示す８２５－２領域のＵフラグが１にセットされたユーザＴＳＳとなる。stctxの次の命令がｕＴＳＳ０の再開時に最初に実行される命令となる。このユ

ーザＴＳＳの保存、復帰は、スレッドを識別するタスクレジスタにはいっさい影響しない（タスクレジスタは、割り込みによって中断されたスレッドに対応するシステムＴＳＳが保存される場所を示す。）。

【０１７０】

次にステップ１３０８に進み、テーブルにセットしておいた作業中フラグをクリアし、同一プログラムにおける他のスレッド（たとえばスレッド２）の実行を再開させる。再開処理については、後述する。次にステップ１３０９で、ltr命令によってタスクレジスタを読み出し、スレッドＩＤを特定する。そして、ステップ１３１０で、スレッドテーブルのパラメータに指定された本来の処理を開始する。

【０１７１】

同一プログラムの下で、割り込みなどにより、他のスレッド２（タスクレジスタがシステムＴＴＳ２を示している）がこの保護ドメインのコードを実行する場合も、スレッドテーブルを初期化して、egate命令を実行するまでは同じである。２番目以降のスレッドは、すでに最初のスレッドによって初期化が行われているので、テーブルのスレッドＩＤ０に対応するユーザＴＳＳが１にセットされている。したがって、ステップ１３０２で初期化済みと判断される（１３０２でＹＥＳ）。ステップ１３０５で、スレッドテーブルが作業中にマークされているかどうかを判断する。作業中であれば（１３０５でＹＥＳ）、スレッド２は休眠して（ステップ１３０６）、ステップ１３０５を繰り返し、テーブルに対する作業の完了により実行が再開されるのを待つ。

【０１７２】

ステップ１３０５でテーブルが作業中でなければ（１３０５でＮＯ）、スレッドＩＤ０に対応するユーザＴＳＳ０を復帰し、ｕＴＳＳ０の保存直後の状態（ステップ１５０８）に制御が移る。このとき、ｕＴＳＳ０に保存されたデータ保護状態が復帰される。しかし、このままでは、これからスレッド１を実行するのかスレッド２を実行するのか区別がつかない。そこで、ステップ１３０９でタスクレジスタを読み出し、自分のスレッドのスレッドＩＤを特定して、パラメータを読み出し、ステップ１３１０で、必要な処理（例えばシグナルハンドラ）の実行

を行なう。ｕＴＳＳの復帰ではタスクレジスタは更新されないので、スレッドは自己のスレッドＩＤを正しく取得できる。

【０１７３】

一度実行を開始してデータ保護属性を共有したスレッドが、データ暗号化属性を変更する場合には、ＴＳＳ１のスレッドを例にとれば、スレッドＩＤ　ＴＳＳ１に対応するテーブルのパラメータフィールドに必要な情報が含まれたパラメータブロックを書き込み、ｕＴＳＳ１をユーザＴＳＳとして指定して上述のecall命令を発行してからステップ１５０３へ制御を移す。ＴＳＳ１の作業中フィールドに作業中フラグを書き込み、他のスレッドに対して休眠を要求し、他のスレッドが休眠状態になるとデータ暗号化属性の変更を開始し、データ暗号化属性の更新が完了すると、ｕＴＳＳ１へと制御を戻す。

【０１７４】

（第２実施形態）

第１実施形態では、データの暗号化鍵などのデータ暗号化属性を全てマイクロプロセッサ内部のレジスタファイル２５３に格納していた。この方法では、暗号化属性の種類が保護属性レジスタの数を越えて増えると、保護属性レジスタの入れ替えを余儀なくされ、プログラミングが繁雑になるとともに処理性能の低下につながる。一方、保護属性レジスタの数を増やすとコンテキスト切替の際にメモリに保存するデータの量が増え、やはり処理性能が低下してしまう。プロセス間通信の暗号化を、マイクロプロセッサの暗号化機能によって行う応用を考えれば、通信相手のプロセス毎に別の暗号化鍵を使うのが普通なので、暗号化属性の種類が多くなることは容易に想像できるので、それに対する対策が必要となる。

【０１７５】

このような場合、一般に情報を直接マイクロプロセッサ内のレジスタに保持するのではなく、外部のメモリ（たとえばメインメモリ２８１）上のテーブルに書き込んでおき、必要な部分だけを動的にプロセッサ内部に読み込む手法が知られている。しかし、秘密を守るという観点からは、この方法は危険である。なぜなら、メモリ上のテーブルが攻撃者によって別のデータ（たとえば解読可能な鍵）が格納されたテーブルにすりかえられてしまうおそれがあるからである。

【０１７６】

そこで、本発明の第２実施形態では、秘密保護に必要な情報の中で、動的に読み込まれる部分ごとに、あらかじめマイクロプロセッサによる署名を付加しておく。そして、この部分が外部のメモリテーブルからマイクロプロセッサ内部に読み込まれる時に、署名を検証する。この方式により、毎回のコンテキスト切替の際のオーバヘッドを減少させると同時に、すり替えに対する安全性を保証している。

【０１７７】

図１５は、本発明の第２実施形態にかかるマイクロプロセッサのレジスタファイル２５３の構成を示す。第２実施形態のマイクロプロセッサは、内部レジスタに、各プログラムから参照されるデータのための暗号化属性をあらかじめ格納する４個の暗号化属性レジスタＣＹ０～ＣＹ３と、暗号化属性を特定するための情報を格納する６０ワードのＣＴ０～ＣＴ５９を有する。ＣＹ０～ＣＹ３は、それぞれが１０ワード、ＣＴ０～ＣＴ５９は、それぞれ４ワードの大きさを持つ。

【０１７８】

ＣＹ０～ＣＹ３はレジスタに暗号鍵フィールドを持っているが、ＣＴ０～ＣＴ５９は暗号化鍵フィールドを持たず、代わりに鍵へのオフセットアドレスを持っている。ＣＴ５９のオフセットアドレスは１５２１－４で示されている。オフセットは鍵格納領域の先頭からアドレスを示し、鍵格納領域の先頭は暗号化属性レジスタＣＹ０で定義される。

【０１７９】

暗号化属性情報特定レジスタＣＴ５９の開始アドレス１５２１－１は暗号化対象領域の先頭を指定し、サイズ１５２１－５はその長さを指定する。キーオフセット１５２１－６は、ＣＹ０で指定された鍵格納領域の先頭からの鍵へのオフセットを示し、そのアドレスは鍵の値とハッシュからなる鍵エントリが格納される。Ｅフラグ１５２１－４は、そのレジスタが有効かどうかを示し、Ｄフラグ１５２１－３はデバッグ状態かどうかを示している。暗号化アルゴリズムと鍵の長さを指定するフィールドはそれぞれ１０２１－２、１０２１－７である。このレジスタはＳａｌｔ１０２１－８を有する。Ｓａｌｔについては、後述する。

【０１８０】

図１６は、マイクロプロセッサ内部のＣＹ０レジスタおよび暗号化属性情報特定レジスタＣＴ０と、これらのレジスタで定義される外部メモリ上の鍵格納領域の関係を示した図である。暗号化属性レジスタＣＹ０によって定義されるメモリ上の鍵格納領域１６０１には、各キーエントリ（鍵エントリ）１６０２、１６０３、…がある。各キーエントリは、鍵の値を示すフィールド１２０２－１、１２０３－１と、この鍵値およびその他の情報についての書名１２０２－２、１２０３－２を有する。

【０１８１】

暗号化属性レジスタＣＹ０～ＣＹ３の初期化時には、暗号化属性情報特定レジスタＣＴ０～ＣＴ５９の全てのレジスタの有効フィールド１５２１－４（図１５）がクリアされ、これらのレジスタで指定された暗号化機能が無効となっている。一方、ＣＹ０～ＣＹ３は有効で、論理アドレス空間のすべてを、初期化時に選択した乱数で暗号化するよう設定されている。

【０１８２】

ＣＴ０～ＣＴ５９の暗号化機能を使う場合は、まず論理アドレス上の鍵の格納領域とその領域のためのマスターキーを選択し、それをＣＹ０に設定して、ＣＹ０レジスタの暗号化機能を有効とする。マスターキーはマイクロプロセッサの例外処理部１３１の乱数生成機構２５２によって生成されたものでも、予めプログラムに埋め込まれた固定鍵でもよい。

【０１８３】

次に、外部メモリ上の鍵格納領域１６０１において、レジスタＣＴ０で使う鍵の格納場所（鍵エントリ）１６０３を、暗号化属性レジスタＣＹ０に設定したアドレス領域の中で決定して、ＣＴ０レジスタのオフセットフィールド１５８０－６（図１５）に書き込む。また、暗号化アルゴリズムと鍵の長さを選択して１５８０－２，１５８０－７に書き込む。

【０１８４】

そして、鍵の値を選択して、鍵値のフィールド１６０３－１に書き込む。必要に応じてＳａｌｔ１５２１－８に乱数を書き込んでから、有効ビットＥ１５２１

－４を１に設定する。

【０１８５】

マイクロプロセッサは、保護テーブル管理ユニット２３３（図２参照）を有する。保護テーブル管理ユニット２３３は、予め定められたアルゴリズムによって、定められた署名の計算対象について、マイクロプロセッサの秘密鍵Ｋｓに基づく署名を計算する。署名した値を、鍵の値（キーバリュー）に続く署名領域１６０３－２に書き込む。このとき、保護テーブル管理ユニット２３３は、鍵と署名の格納領域１６０３がＣＹ０の範囲に入っているかどうか、およびＣＹ０の暗号化機能は有効かどうかを確認する。鍵領域１６０３がＣＹ０の範囲外である場合、あるいはＣＹ０の暗号化鍵が０に設定されていて暗号化機能が有効でない場合には、後述するように、マイクロプロセッサの例外処理ユニット２５５が秘密鍵保護例外を発生させる。ＣＴ０～ＣＴ５９の安全性はＣＹ０の安全性に依存するので、ＣＹ０の暗号化鍵は単に乱数を選ぶだけでなく、使用する暗号化アルゴリズムについて安全なことを確認することが望ましい。

【０１８６】

署名の計算対象には、暗号化鍵の値１６０３－１、ＣＴ０レジスタ１５８０の内容、および暗号化鍵の論理アドレスが含まれる。ＣＴ０レジスタ１５８０にはＳａｌｔが含まれているので、適切な頻度でＳａｌｔの値を変更すれば、攻撃者が過去に使われた暗号化された鍵エントリを再利用して、鍵エントリをすりかえることは、たとえＣＹ０のその他のフィールドの値が同一であったとしてもきわめて困難である。もちろん、全く属性値の異なる鍵エントリを利用した場合の、鍵エントリのすり替えは暗号学的に不可能である。

【０１８７】

データの読み込みの際には、論理アドレスと、暗号化属性レジスタＣＹ０～ＣＹ３および暗号化属性情報特定レジスタＣＴ０～ＣＴ５９を比較し、暗号化属性を決定する。読み込み対象の論理アドレスがＣＴ０で指定された領域ならば、オフセットの示す暗号化鍵を鍵キャッシュに読み込む。コード暗号化鍵・署名検証ユニット２５７は、読み込まれた鍵の値と、ＣＴ０の内容と、鍵のアドレスとから、マイクロプロセッサの公開鍵Ｋｐによって署名を検証する。検証の結果がエ

ラーであれば、秘密保護例外を発生して処理を停止する。検証の結果が正しければ、データＴＬＢ１４１の復号化ユニット２３７において、取り出された鍵でデータを復号化し、復号化されたデータがデータキャッシュ２１８に読み込まれる。

【０１８８】

＜再開不能ビット＞

検証に失敗して秘密保護例外が発生した場合、コンテキストの再開可能ビットが０にクリアされて、以後そのコンテキストの再開は不可能となる。このような違反が発生するのは、何らかの攻撃者によって、秘密保護のためのデータが破壊されている可能性が高いため、これに続く実行を禁止して、秘密情報の流出を防いでいる。

【０１８９】

なお、ここでは暗号化属性レジスタＣＹ０～ＣＹ２に、暗号化対象領域の論理アドレスの先頭とサイズを設定する場合を例にとったが、この他にもページテーブルや、セグメントディスクリプタのようなテーブル形式を利用したメモリ暗号化属性の管理方式が考えられる。これらの方式にも本発明が適用されることは言うまでもない。

【０１９０】

第２実施形態においては、データの暗号化属性の決定には必要のない、サイズの大きな鍵情報をプロセッサ外部のテーブルに格納して、プロセッサ内部のコンテキスト情報のサイズを低減すると同時に、外部のテーブルに格納される鍵情報の一つ一つにハッシュを付加し、読み込み時に検証することで、プロセッサ外部に格納される情報のすりかえによる攻撃を防止している。

【０１９１】

ここで、署名は、マイクロプロセッサの暗号化属性情報特定レジスタの値を含むデータから生成されているため、暗号化属性情報特定レジスタが有するＳａｌｔの値を変えることにより、過去にプロセッサ外部のメモリに書き込んだ鍵エントリを無効化して、過去に使われた鍵エントリの流用を防止することができる。このような署名方法を使用せずに、たとえば、アドレス範囲属性と鍵の値だけか

ら署名を生成するならば、アドレス範囲さえ一致すれば過去に使われた鍵エントリを外部のメモリ上ですりかえる攻撃が可能となってしまう。その一方で、もし署名生成の対象に汎用レジスタが含まれていれば、汎用レジスタの値が変わる都度、署名を生成しなおさなければならなず、性能の低下を招く。

【０１９２】

本発明では、暗号化属性情報特定レジスタの属性値と、Ｓａｌｔ値さえ書き換えなければ、署名を再計算する必要はない。実際の使用では、暗号化属性情報特定レジスタの設定頻度はそれほど高くないと考えられるので、性能の低下を押さえつつ、鍵の値のすりかえを防止することができる。

【０１９３】

第２実施形態にかかるマイクロプロセッサにおいては、すべての暗号化属性をマイクロプロセッサのレジスタ上に格納する場合と比較して、コンテキスト情報が著しく小さくなり、コンテキスト切替の際の暗号化や署名に必要な計算量や、コンテキストを保存するためのメモリトラフィックを減少させて、プロセッサの性能を向上させることができるのである。

【０１９４】

署名をメモリ側でなく、レジスタの側においても同様にすり替えを防ぐことができるが、一般に非対称鍵方式を使った安全な署名には、大きなデータ長を必要とするので、レジスタの側にSalt値を置き、メモリ上に署名を置く方がコンテキスト情報を小さくする効果が大きい。

【０１９５】

本発明では、暗号化エントリを有効にする際に、対応するメモリ上の鍵格納領域１６０１が暗号化されていること（すなわちエントリに有効な鍵があること）を検証するので、暗号化鍵の保護をさらに確実にしている。

【０１９６】

＜ページテーブル形式＞

また、データ保護のテーブル形式を、仮想記憶の管理に使われるページテーブルと同様のアドレス単位ごとに階層化された形式とし、各々のエントリ毎に署名を付加したものにすることも可能である。ページテーブル形式は理論的にはペー

ジ毎に異なる暗号化属性を与えることができる。さらに、マイクロプロセッサ内部に保持するデータは、ページテーブルの先頭アドレスとＳａｌｔだけでよく、コンテキスト情報の量を小さくすることができるという効果がある。

【０１９７】

混乱を避けるため以下、マイクロプロセッサ内部でデータ暗号化に使われるテーブルを、暗号化定義ページテーブルと呼ぶことにする。暗号化定義ページテーブルは、外部メモリの仮想記憶ページテーブルとは異なり、ユーザプログラムの論理メモリ空間上に置かれる。

【０１９８】

また、暗号化定義ページテーブルの各エントリに付与する署名が、単純にプロセッサの暗号化定義ページテーブルレジスタ情報およびエントリに含まれる情報だけから生成される場合、任意のエントリが署名を含んだ状態で交換されたとしても、それを検出することはできない。プロセッサ側に置かれる情報、すなわち暗号化定義ページテーブルレジスタとそれに含まれるＳａｌｔが全てのエントリについて共通となってしまうからである。この問題は、暗号化定義ページテーブルの各エントリの署名の計算領域に、エントリが置かれる論理アドレスを含めることによって（すなわち、エントリごとに異なる値が署名に含まれるようにすることによって）、解決される。

【０１９９】

なお、ページテーブルの初期化は、あらかじめキーバリュー（鍵値）などの属性をメモリのテーブルに書き込んでおき、そのアドレスをマイクロプロセッサ内部の暗号化定義ページテーブルレジスタに設定し、Ｓａｌｔの値を同じレジスタに書き込んでこのレジスタの有効フラグをセットする。有効フラグのセットにより、ページテーブルの各エントリに対する署名がマイクロプロセッサにより計算され、外部メモリ上に書き込まれる。階層化されている場合は階層を順次探索して計算が行なわれる。この処理は、エントリ数に比例した時間を要することに留意しなければならない。

【０２００】

＜デバッグについて＞

実行コードと、その処理対象であるデータの双方が保護されるとしても、そのような機能の開発途上のデバッグ作業では、暗号化することなく平文でデバッグできることが開発効率上望ましい。

【０２０１】

デバッグに関する設定は２箇所で行う。ひとつは、暗号化コードのページテーブルエントリに設けられた実行コードのデバッグビット３０７－ｊ－Ｄ（図４参照）であり、もうひとつは、図１５に示すように、データの暗号化鍵属性を指定するレジスタに設けられた暗号化制御ビット（ＣＹ３においては１５１７－３、ＣＴ５９においては１５２１－３）である。後者の第１実施形態における対応ビットは、図７に示す暗号化属性レジスタＣＹ３のビット７１７－３である。

【０２０２】

ページテーブルエントリに設けられたデバッグビット３０７－ｊ－Ｄがセットされている時、マイクロプロセッサの実行コード復号化ユニット２１２（図２参照）は迂回され、エントリに対応するメモリ上におかれた実行コードは、平文状態のまま解釈され、実行される。デバッグ状態でコードを実行している間は、プロセッサのデバッグ機能、たとえば第１実施形態と関連して述べたステップ実行機能やデバッグレジスタが有効化される。

【０２０３】

ここで、ページテーブルのデバッグビットは何ら保護を受けていないが、悪意のあるユーザが、暗号化された実行プログラムのおかれているページテーブルエントリのデバッグビットをセットしたとしても、暗号化された実行プログラムが復号化されずに命令フェッチ／デコード機能２１４に取り込まれるだけで、正常実行はできないので、プログラムの秘密が洩れるおそれはない。

【０２０４】

デバッグ実行中は、例外発生時のコンテキスト保存も平文のまま行われる。したがって、コンテキストを暗号化する乱数Ｋｒをコード暗号化鍵Ｋcodeで暗号化した値$E_{Kcode}[Kr]$８３３は、平文のＫｒにおきかえられる。この場合コンテキスト情報がコードの暗号鍵と一致するかどうかのチェックを行うことはできないが、デバッグの場合にはこのような攻撃の危険を考慮する必要がない。

【０２０５】

実行コードの演算処理対象であるデータの暗号化は、第１実施形態においては図７に示す暗号化制御ビット７１３－３によって、第２実施形態においては図１５に示すＣＹ３の暗号化制御ビット１５１７－３、およびＣＴ５９の暗号化制御ビット１５２１－３によって制御される。暗号化制御ビットが０の時は、デバッグ状態でもデータは暗号化状態で処理される。暗号化制御ビットが１にセットされている時は、デバッグ状態の場合に限りデータは平文のまま処理される。

【０２０６】

本発明ではデータについて、デバッグ状態で暗号化属性のエントリごとに、暗号化と非暗号化を選択できるようにすることで、別のプログラムやデバイスとの間で行われる暗号化されたデータ送受信のデバッグと、プログラムの内部的なデータの平文状態でのデバッグとを、容易に両立させている。

【０２０７】

特にコードについてはコードを暗号化せず平文として、ページテーブル上のデバッグフラグをセットするだけで、プログラム自体を再コンパイルすることなくデバッグが行なえる利便性を提供している。

【０２０８】

処理対象データについては、他のプロセスとの通信に使われる領域は、自プロセスはデバッグモードだったとしても、相手のプロセスがデバッグ状態でない限り平文にしておくことはできない場合がある。そこでコードがデバッグモードで動作している場合のみ有効になる暗号化フラグをデータの暗号化属性に設け、このフラグによって暗号化状態を制御できるようにしている。

【０２０９】

コードのデバッグには、コード自体の暗号状態と、コードを暗号化する鍵の暗号化状態とで、４通りの組み合わせが考えられる。上述した例は、コード暗号鍵もコードも平文の場合の組み合わせ例である。

【０２１０】

他のモード（組み合わせ）も考えられる。第１が、コード暗号化鍵エントリもコード自体も暗号化された状態で、デバッグモードでコードを実行できるモード

である。第２が、コード暗号化鍵エントリは平文の状態で、コード自体は暗号化された状態で、デバッグモードでコードを実行できるモードである。第３が、コード暗号化鍵エントリは暗号化状態、コードは平文状態の場合である。

【０２１１】

デバッグの目的からはこれらのモードにはあまり意味を持つものではないが、必要に応じてこれらのモードを備えてもよい。

【０２１２】

この際に注意しなければならないのは、第１のケースで、暗号化されたコード暗号化鍵エントリに対応する暗号化された実行コードをデバッグモードで実行可能とする場合は、ユーザが暗号化制御のためのビットを操作するだけで、暗号化鍵を知らない保護されたプログラムを、デバッグモードで実行することができてしまう。

【０２１３】

この場合には、デバッグモードビットを不正に操作されないように、デバッグビットをコード暗号化鍵Ｋcodeとともに、プロセッサの公開鍵Ｋｐで暗号化するなどの対策を施して、真のコード暗号化鍵を知らないユーザにデバッグビットを操作されないように注意しなければならない。

【０２１４】

【発明の効果】

以上説明したように、本発明のマイクロプロセッサによれば、マルチタスク環境下で、実行コードと、そのコードの処理対象であるデータの双方を暗号化して保護することにより、オペレーションシステム、あるいは第三者による不正な解析を防止することができる。

【０２１５】

また、データを暗号化して保存した場合の、暗号化属性の不正な書き換えを防止することが可能になる。

【０２１６】

また、処理対象であるデータの暗号化鍵として、固定鍵ではなく任意の乱数Ｋｒを使用することができ、暗号化されたデータを不正な攻撃から守ることができ

る。

【０２１７】

また、平文状態でデバッグを行い、不具合が発見された場合は、実行コードの暗号化鍵を知っているプログラムベンダにエラーをフィードバックさせることができる。

【０２１８】

さらに、暗号化属性情報など、秘密保護に必要な情報にマイクロプロセッサの署名をつけて外部のメモリ上に保存し、必要な部分だけをプロセッサ内部のレジスタに読み込み、読み込み時に署名の検証を行うことで、マイクロプロセッサのメモリの増大を防止し、コストを押さえることができる。この方式では、読み込み時のすり替えに対する安全性も保障される。

【図面の簡単な説明】

【図１】

本発明の第１実施形態にかかるマイクロプロセッサの基本構成を示す図である。

【図２】

図１のマイクロプロセッサの詳細な構成を示す図である。

【図３】

図１のマイクロプロセッサにおけるページディレクトリと、ページテーブル形式を示す図である。

【図４】

図３に示したページテーブルの詳細と、鍵エントリ形式を示す図である。

【図５】

図１のマイクロプロセッサにおけるデータのインタリーブを示す図である。

【図６】

第１実施形態における実行命令コードのアドレス変換から復号化までの情報の流れを示す図である。

【図７】

第１実施形態にかかるマイクロプロセッサのＣＰＵレジスタの構成を示す図で

ある。

【図８】

第１実施形態におけるコンテキスト保存形式を示す図である。

【図９】

第１実施形態における保護ドメイン切り替えの手順を示すフローチャートである。

【図１０】

第１実施形態における、処理対象であるデータの暗号・復号化の情報の流れを示す図である。

【図１１】

本発明のマイクロプロセッサにおける保護ドメイン内の実行制御を示す図である。

【図１２】

本発明のマイクロプロセッサにおける保護ドメインから非保護ドメインへの呼び出し、分岐を示す図である。

【図１３】

本発明のマイクロプロセッサにおけるデータ暗号化属性共有の手順を示すフローチャートである。

【図１４】

データ保護属性の共有に使用されるスレッドテーブルの図である。

【図１５】

本発明の第２実施形態にかかるマイクロプロセッサのＣＰＵレジスタ構成を示す図である。

【図１６】

第２実施形態にかかるマイクロプロセッサにおける鍵格納領域レジスタおよび暗号化属性情報特定レジスタと、それぞれ対応するメモリ上の鍵格納領域の関係を示す図である。

【図１７】

第２実施形態にかかるマイクロプロセッサにおけるコンテキスト保存形式を示

す図である。

【図１８】

従来技術におけるコンテキスト保存形式を示す図である。

【符号の説明】

１０１　マイクロプロセッサ

１１１　プロセッサコア

１１２　バスインターフェイスユニット（読み出し手段）

１１３　コード・データ暗号化／複号化処理部

１１４　１次キャッシュ

１１５　命令実行部

１２１　命令ＴＬＢ

１３１　例外処理部

１４１　データＴＬＢ

１５１　２次キャッシュ

２１２　コード復号化ユニット

２１９　データ復号化ユニット

２２０　データ暗号化ユニット

２３３　保護テーブル管理部

２５２　乱数発生機構

２５３　レジスタファイル（プロセッサ外部の記憶手段）

２５４　コンテキスト情報暗号／複号化ユニット

２５７　コード暗号化鍵・署名検証ユニット

２８１　メインメモリ（プロセッサ外部の記憶手段））

３０７、５１２　ページテーブル

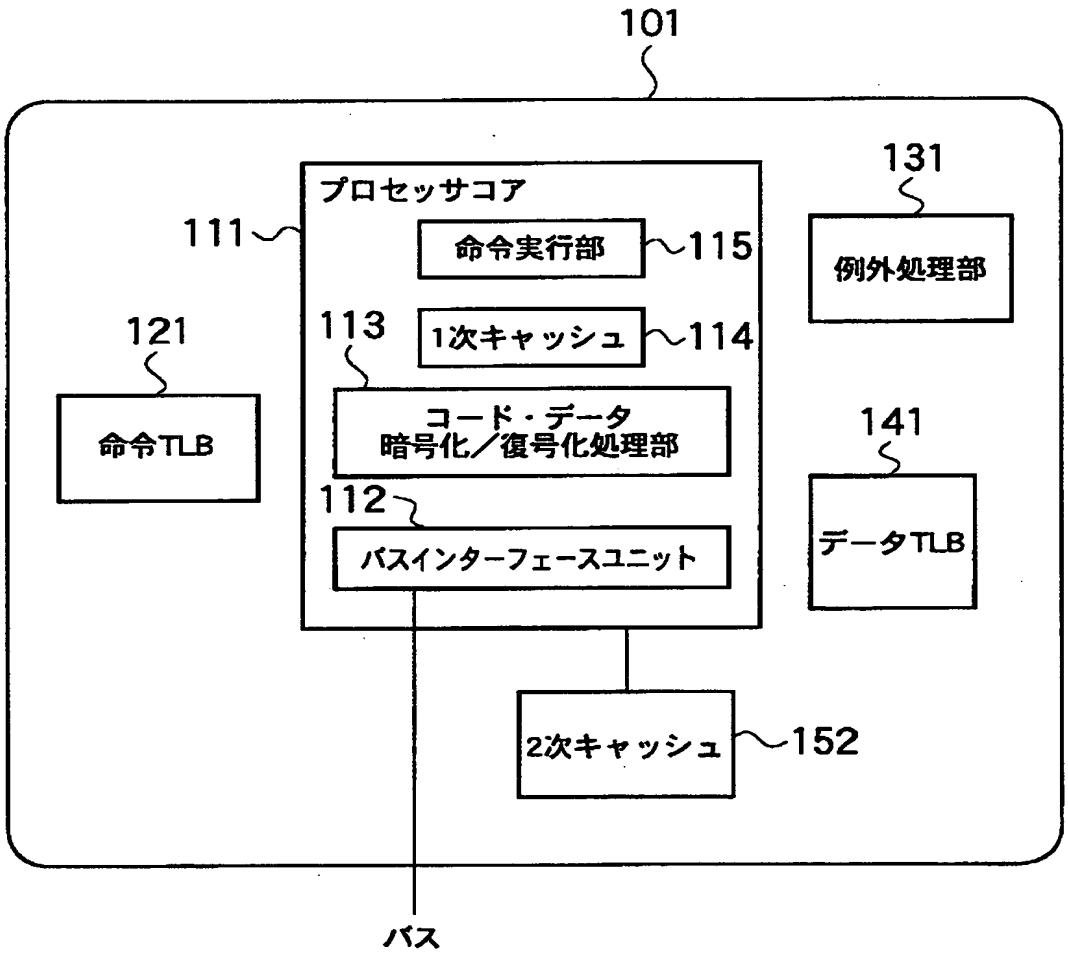
３０９、５１１　キーテーブル

ＣＹ０～ＣＹ３　暗号化属性レジスタ

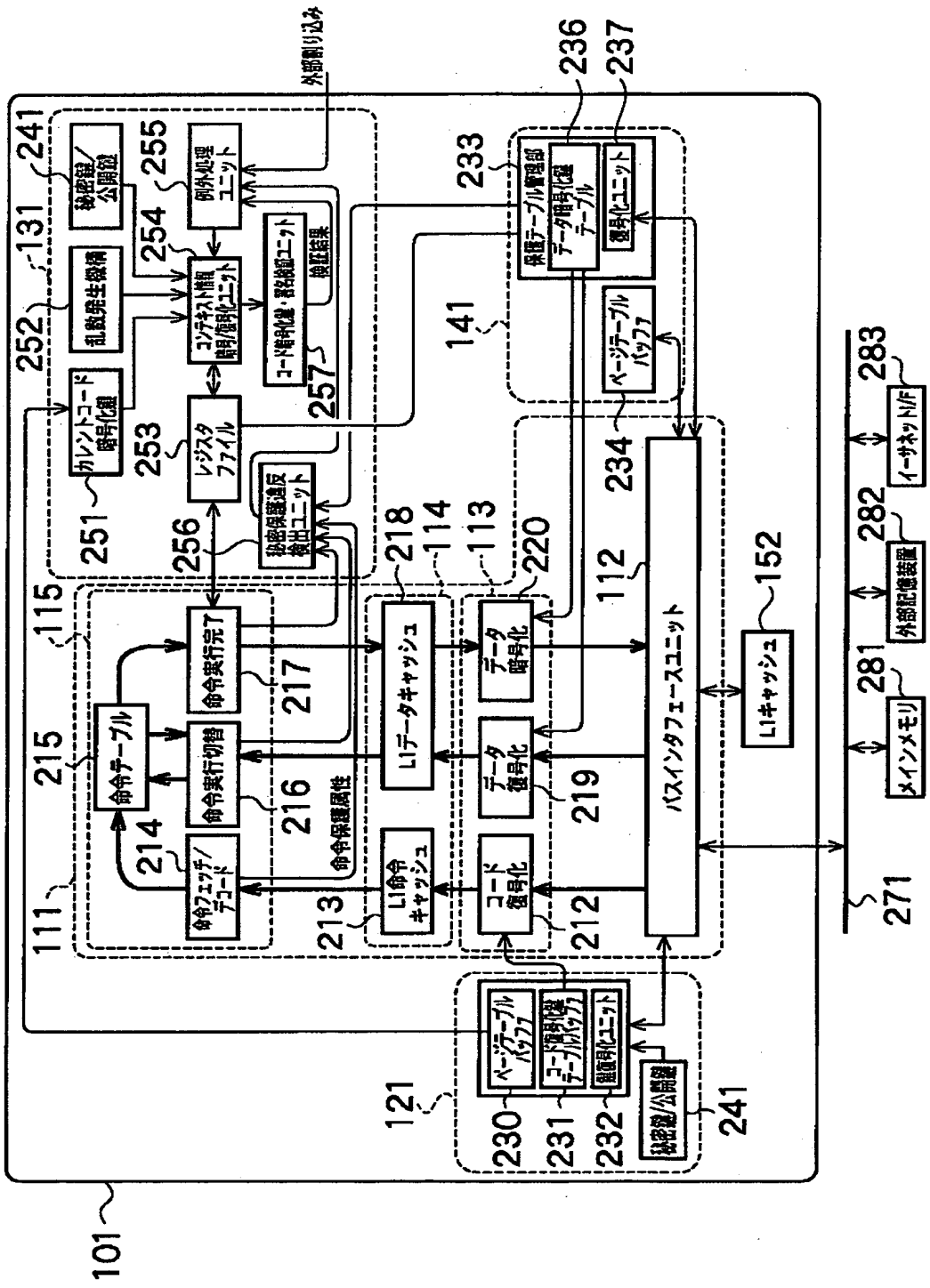
ＣＴ０～ＣＴ５９　暗号化属性情報特定レジスタ

【書類名】　　図面

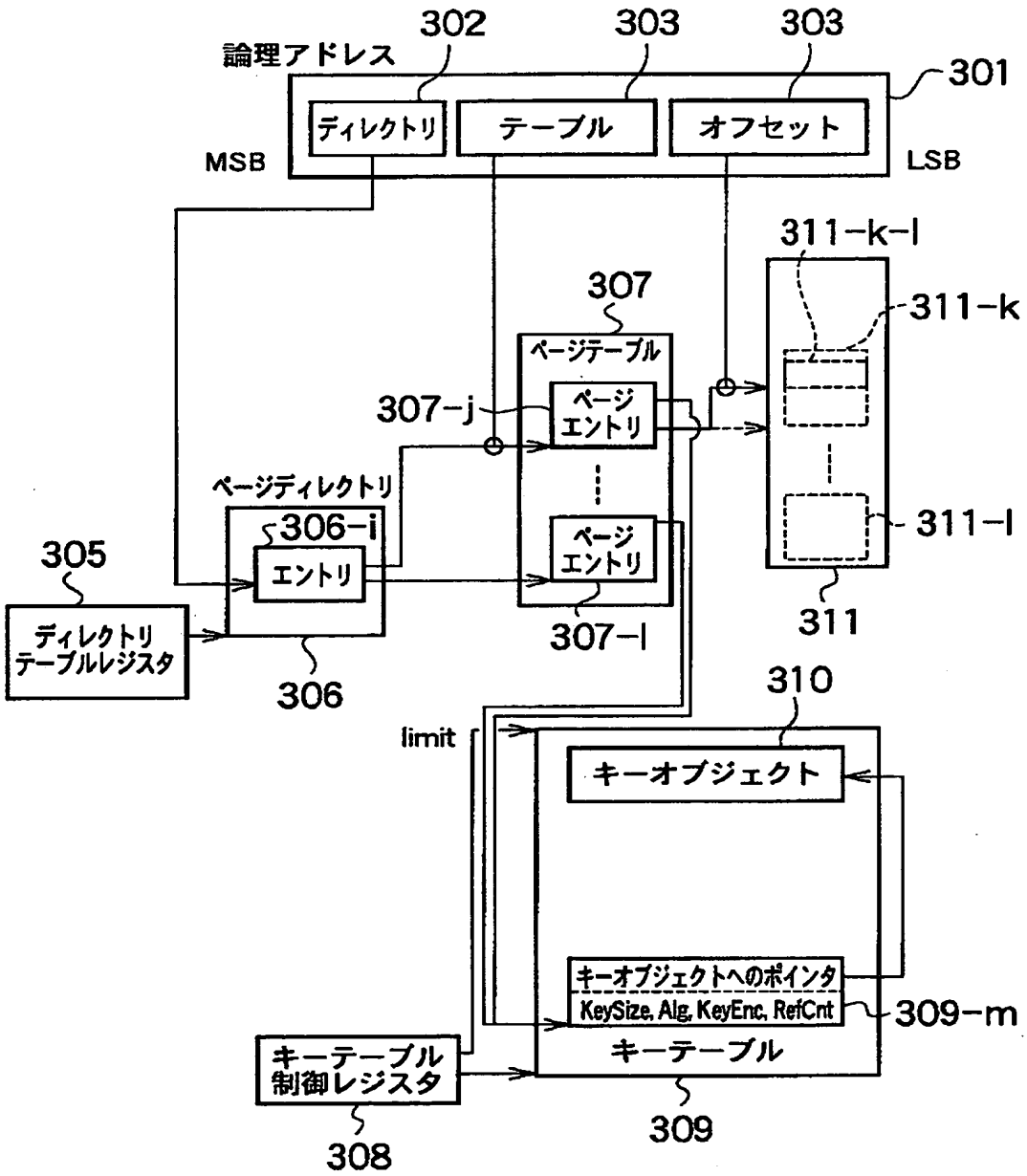
【図１】


101
プロセッサコア
111
命令実行部
115
131
例外処理部
121
113
1次キャッシュ
114
命令TLB
コード・データ
暗号化／復号化処理部
141
112
データTLB
バスインターフェースユニット
2次キャッシュ
152
バス

【図２】

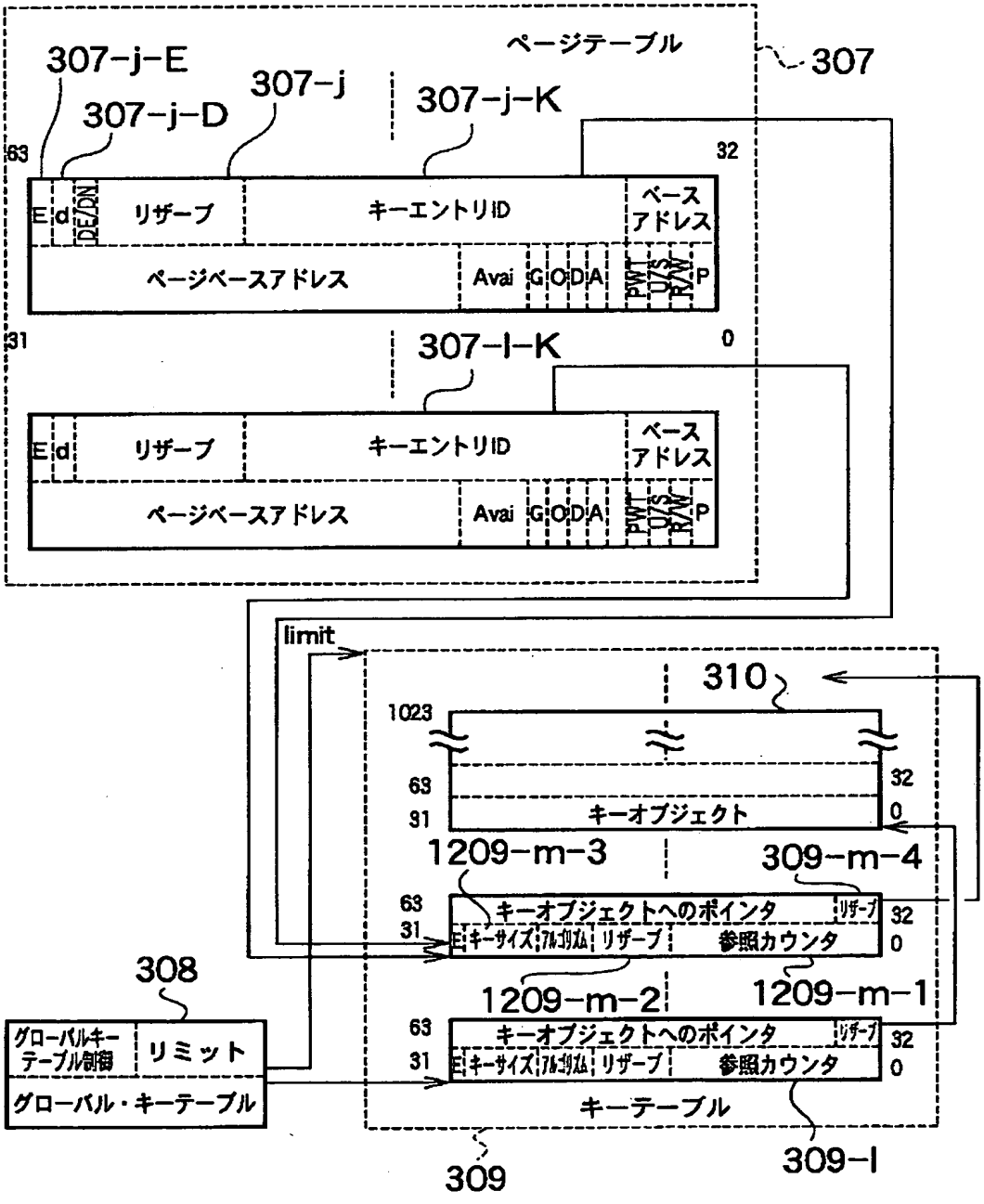

101
111
115
131
141
213
214
215
216
217
218
219
220
230
231
232
233
234
236
237
241
251
252
253
254
255
256
257
271
281
282
283
152
121
112
113
114
外部割り込み
命令テーブル
命令実行切替
命令実行完了
命令フェッチ/デコード
命令保護属性
L1命令キャッシュ
L1データキャッシュ
コード復号化
データ復号化
バスインタフェースユニット
L1キャッシュ
メインメモリ
外部記憶装置
イーサネットI/F
秘密鍵/公開鍵
乱数発生機構
カレントコード暗号化鍵
レジスタファイル
コンテキスト情報暗号/復号化ユニット
例外処理ユニット
コード暗号鍵・認証鍵ユニット
検証結果
秘密保護違反検出ユニット
保護テーブル管理部
データ暗号化テーブル
復号化ユニット
ページテーブルバッファ
ページテーブル
コード暗号化テーブル
コード暗号化テーブルバッファ
暗号化ユニット
秘密鍵/公開鍵

【図３】



ページディレクトリ／ページテーブル形式

【図４】



ページテーブル／鍵エントリ形式

【図５】

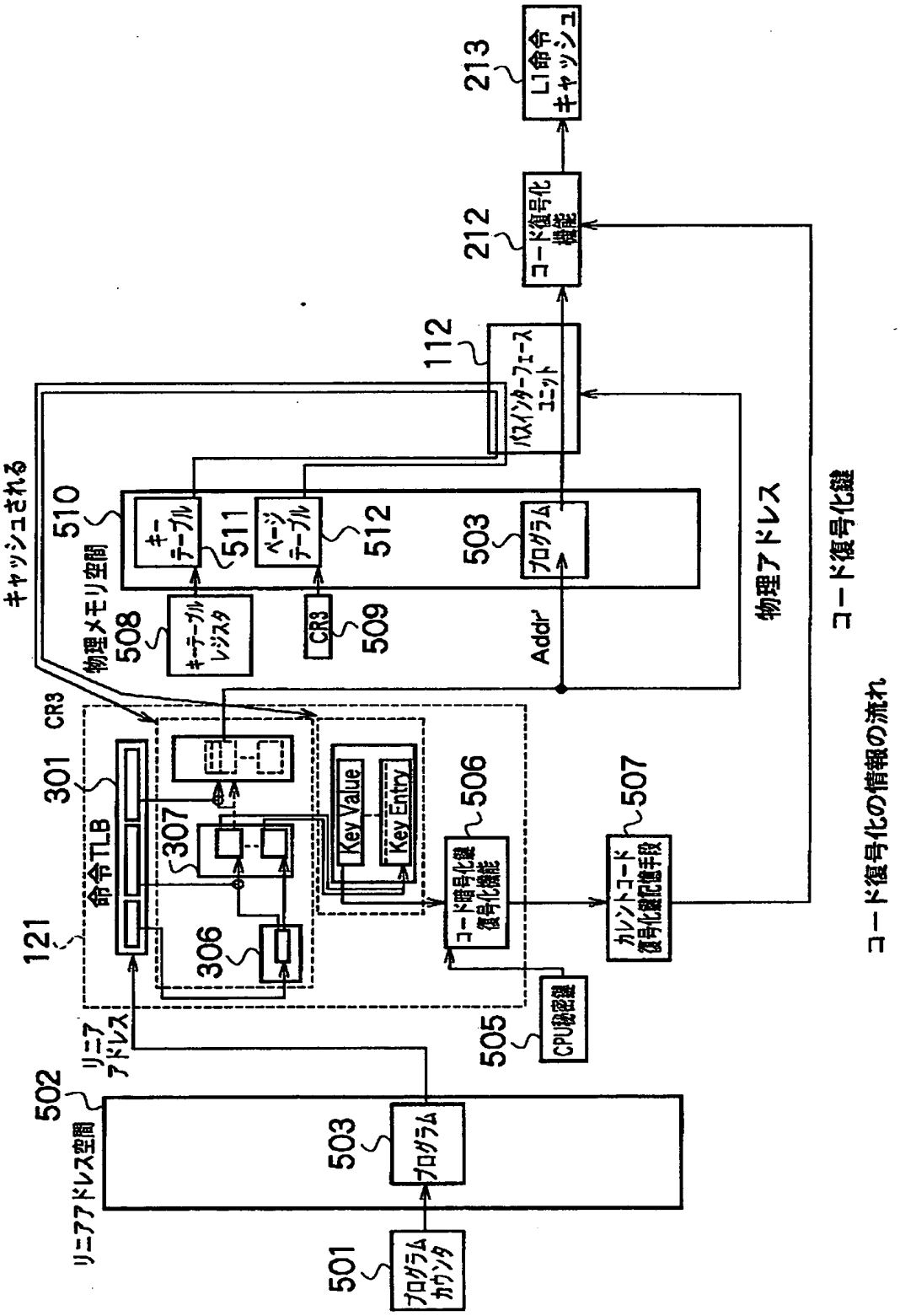
	0	1	2	3
0	A0	A1	A2	A3
1	B0	B1	B2	B3
2	C0	C1	C2	C3
3	D0	D1	D2	D3
4	E0	E1	E2	E3
5	F0	F1	F2	F3
6	G0	G1	G2	G3
7	H0	H1	H2	H3

インタリーブ前

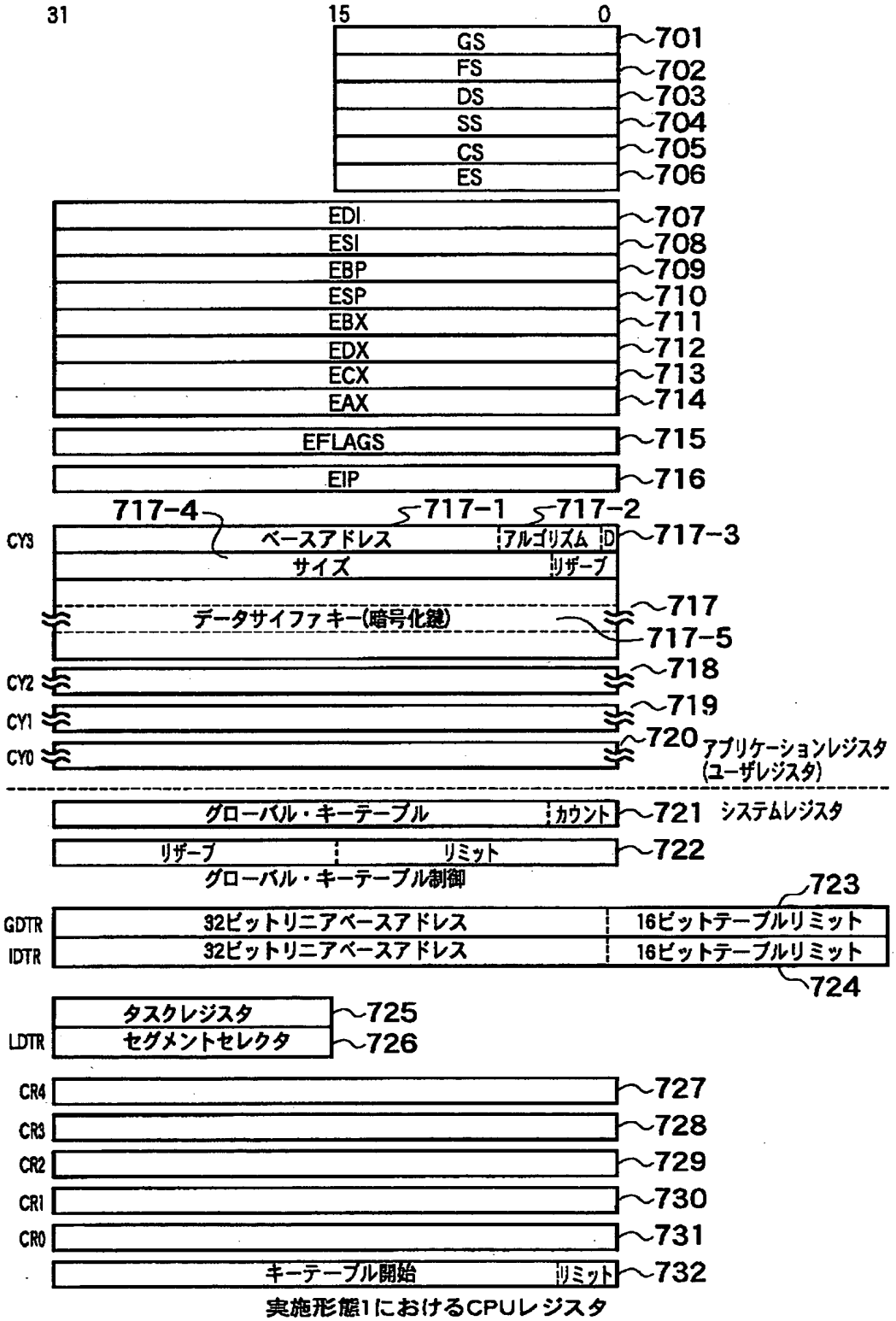
0	A0	B0	C0	D0
1	E0	F0	G0	H0
2	A1	B1	C2	D1
3	E1	F1	G1	H1
4	A2	B2	C2	D2
5	E2	F2	G2	H2
6	A3	B3	C3	D3
7	E3	F3	G3	H3

インタリーブ後

【図６】

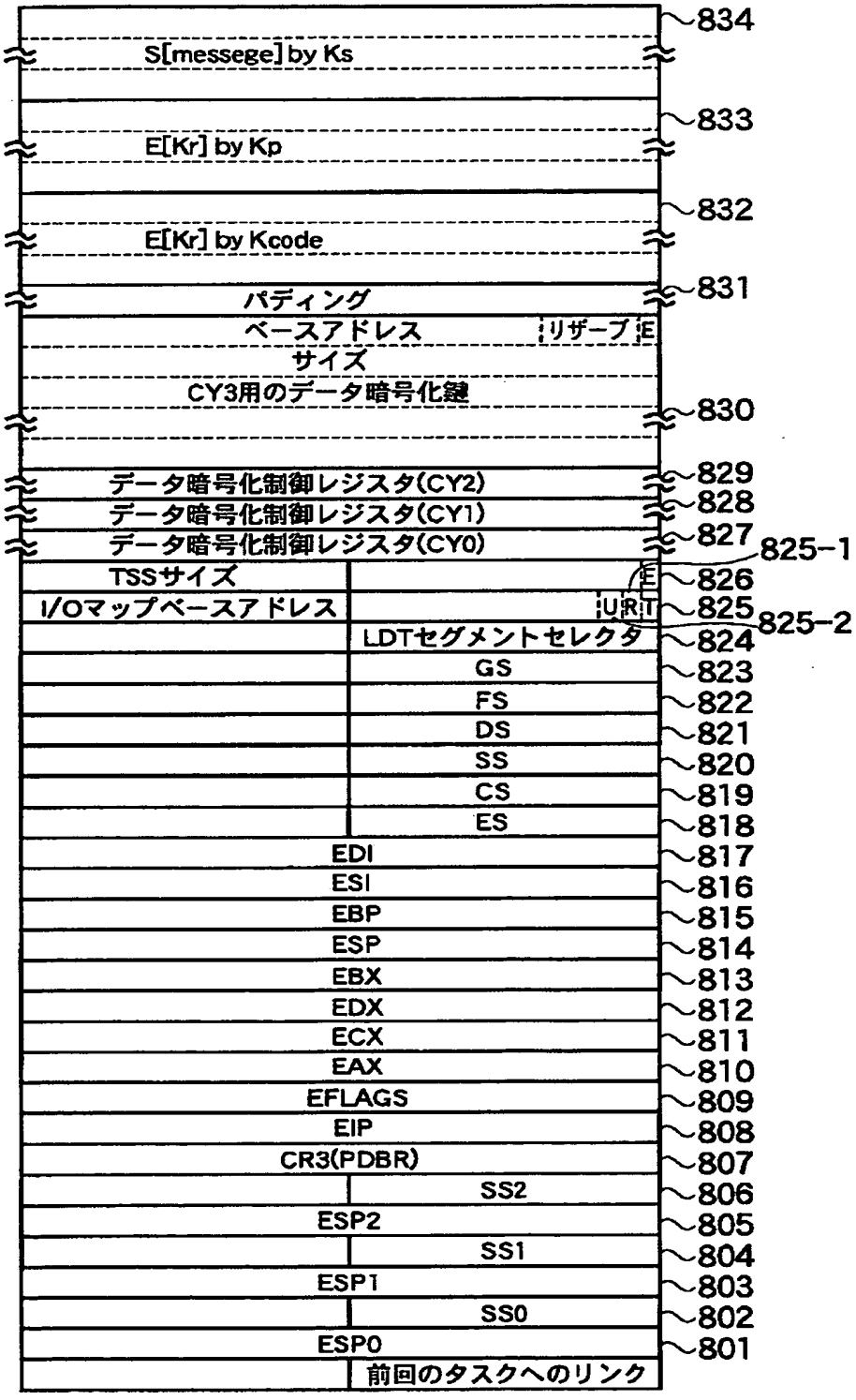

213
L1命令キャッシュ
212
コード復号化機能
112
バスインターフェースユニット
510
キャッシュされる物理メモリ空間
511
キーテーブル
512
ページテーブル
503
プログラム
508
キーテーブルレジスタ
509
CR3
Addr
物理アドレス
コード復号化鍵
コード復号化の情報の流れ
301
CR3
命令TLB
307
306
121
Key Value
Key Entry
506
コード暗号化鍵復号化機能
507
カレントコード復号化鍵記憶手段
505
CPU秘密鍵
リニアアドレス
502
リニアアドレス空間
503
プログラム
501
プログラムカウンタ

【図７】



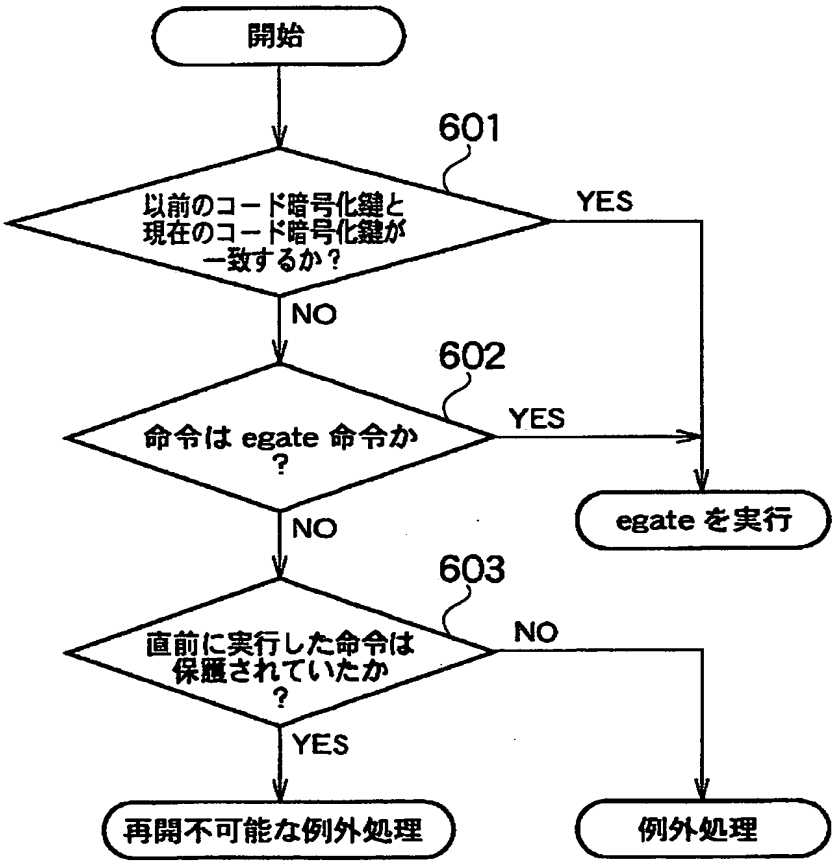
実施形態1におけるCPUレジスタ

【図８】



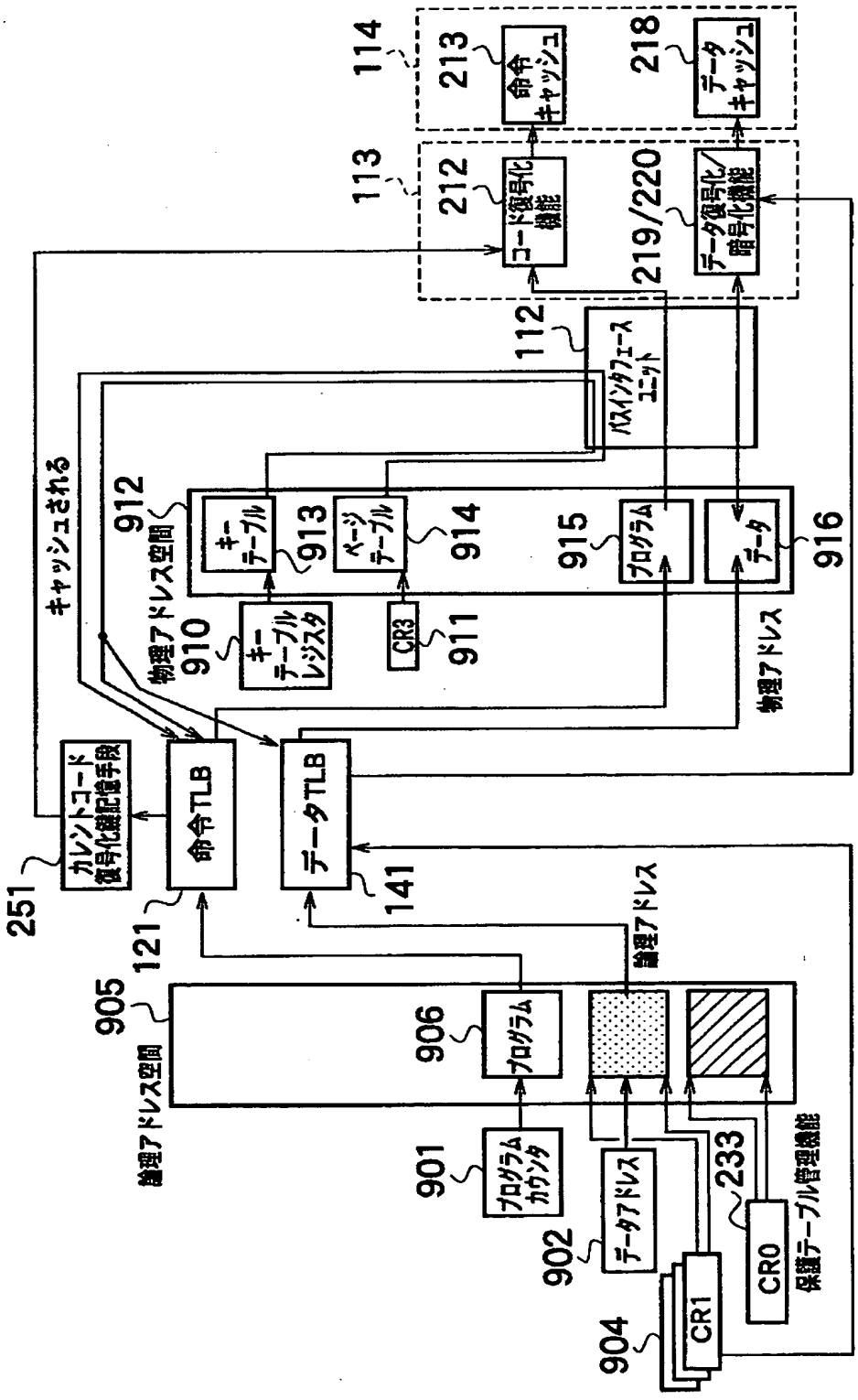
実施形態1におけるコンテキスト保存形式

【図９】



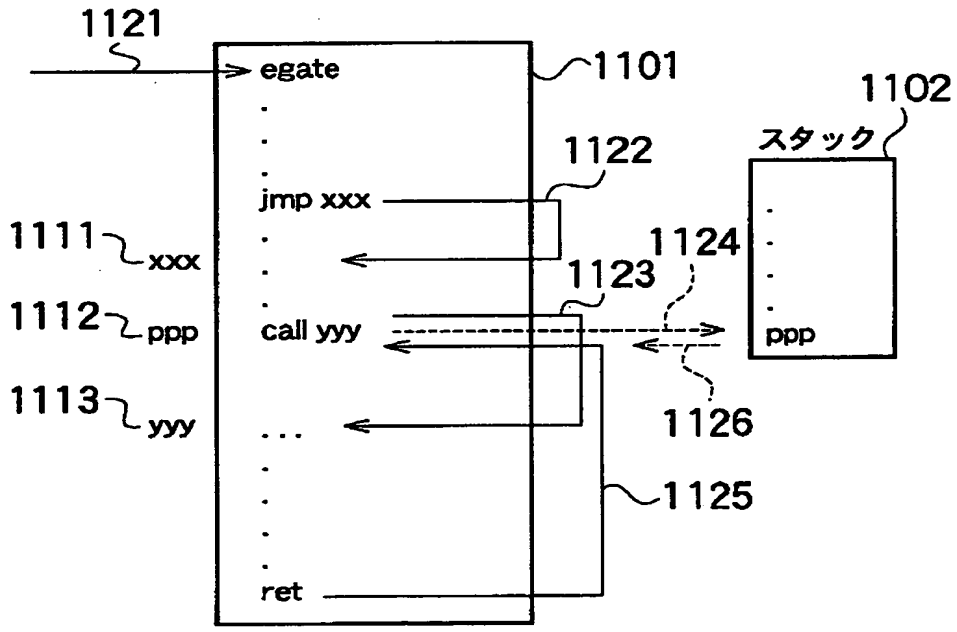
保護ドメイン切り替えの手順

【図１０】



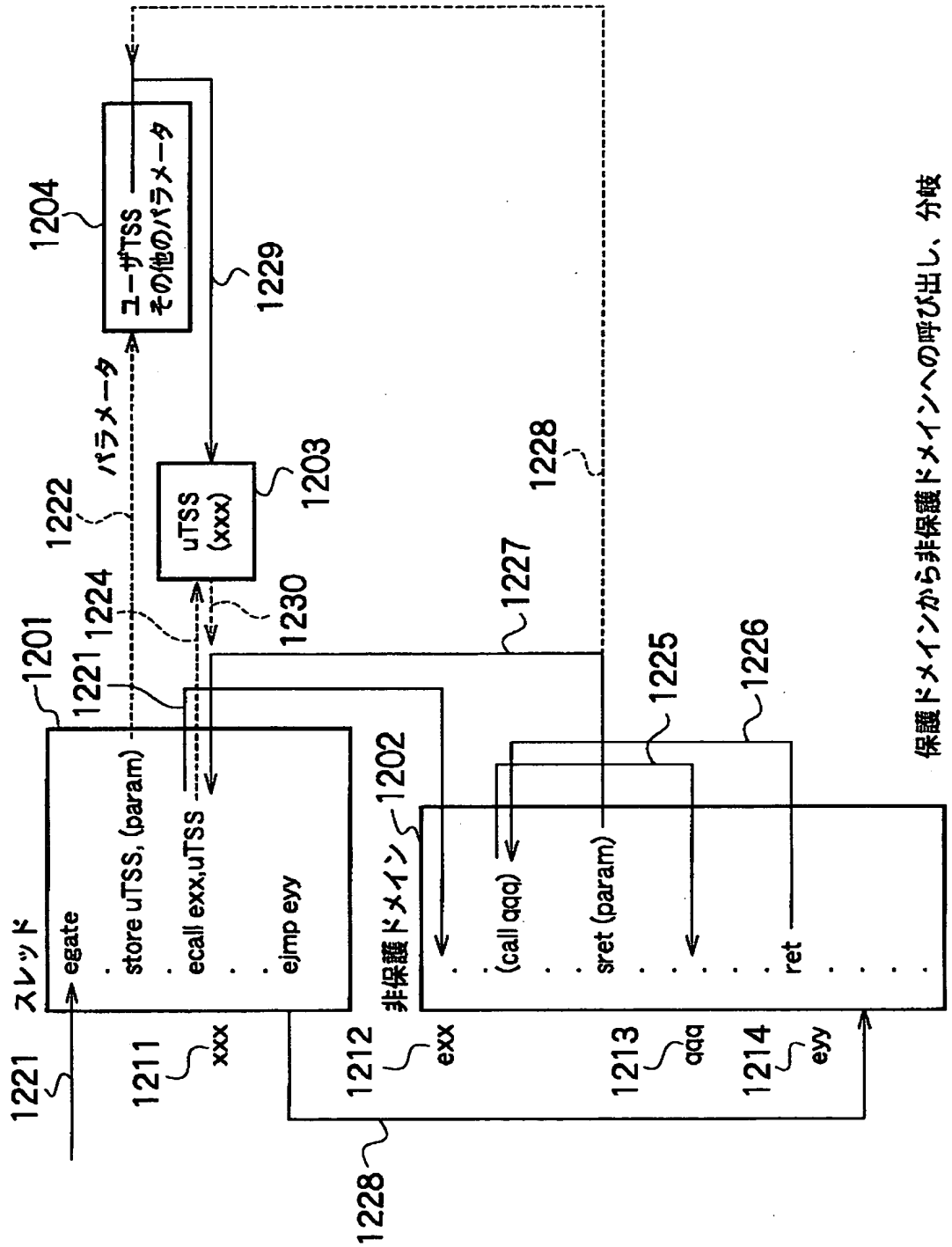
実施例1におけるデータ暗号・復号化の情報の流れ

【図１１】

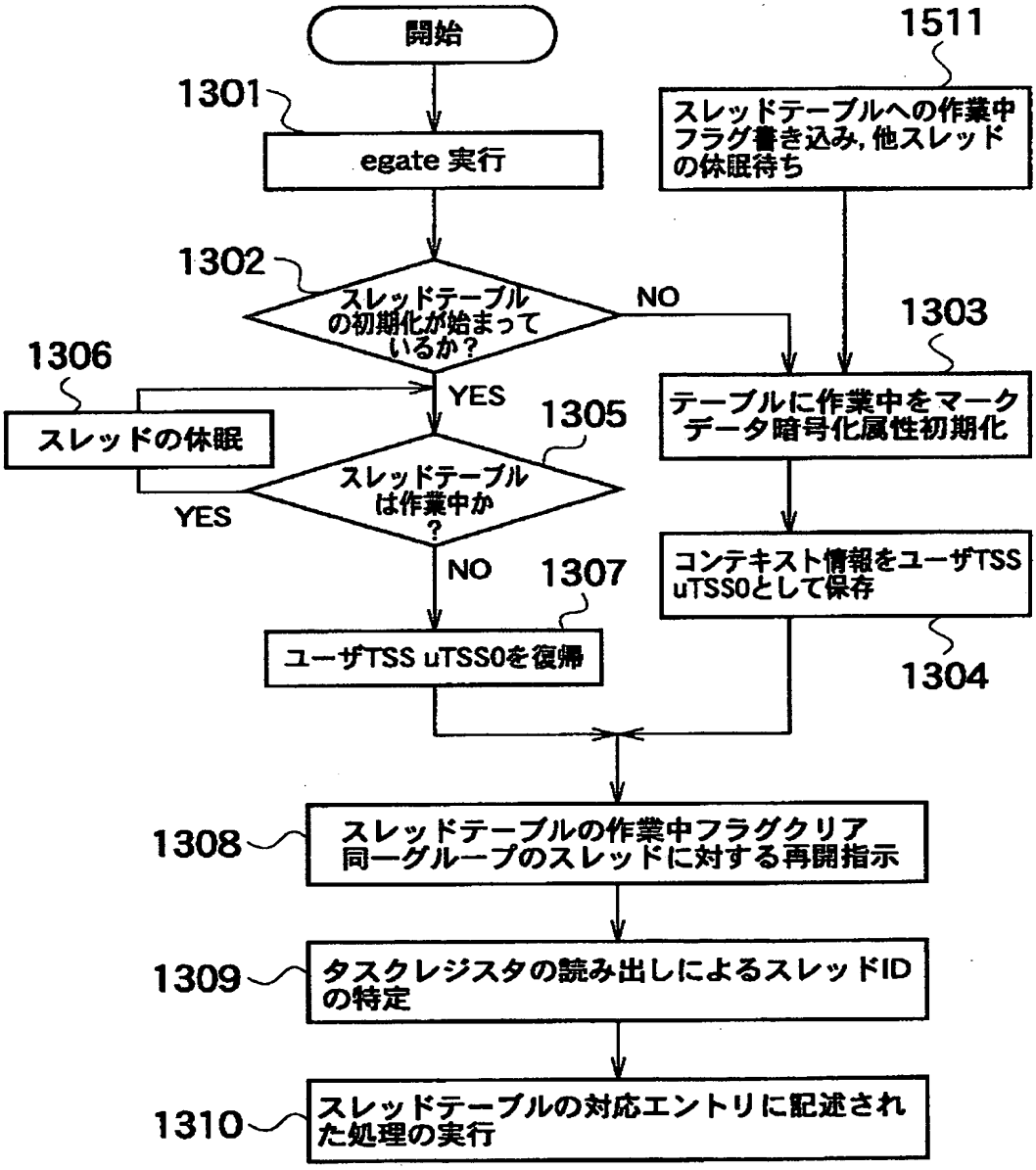


保護ドメイン内の実行制御

【図１２】


1204
ユーザTSS
その他のパラメータ
1229
パラメータ
1222
uTSS
(xxx)
1203
1228
1224
1230
1227
1201
1221
1225
1226
store uTSS, (param)
ecall exx,uTSS
ejmp eyy
スレッド
egate
非保護ドメイン
1202
(call qqq)
sret (param)
ret
保護ドメインから非保護ドメインへの呼び出し、分岐
1221
1211
xxx
1212
exx
1213
qqq
1214
eyy
1228

【図１３】



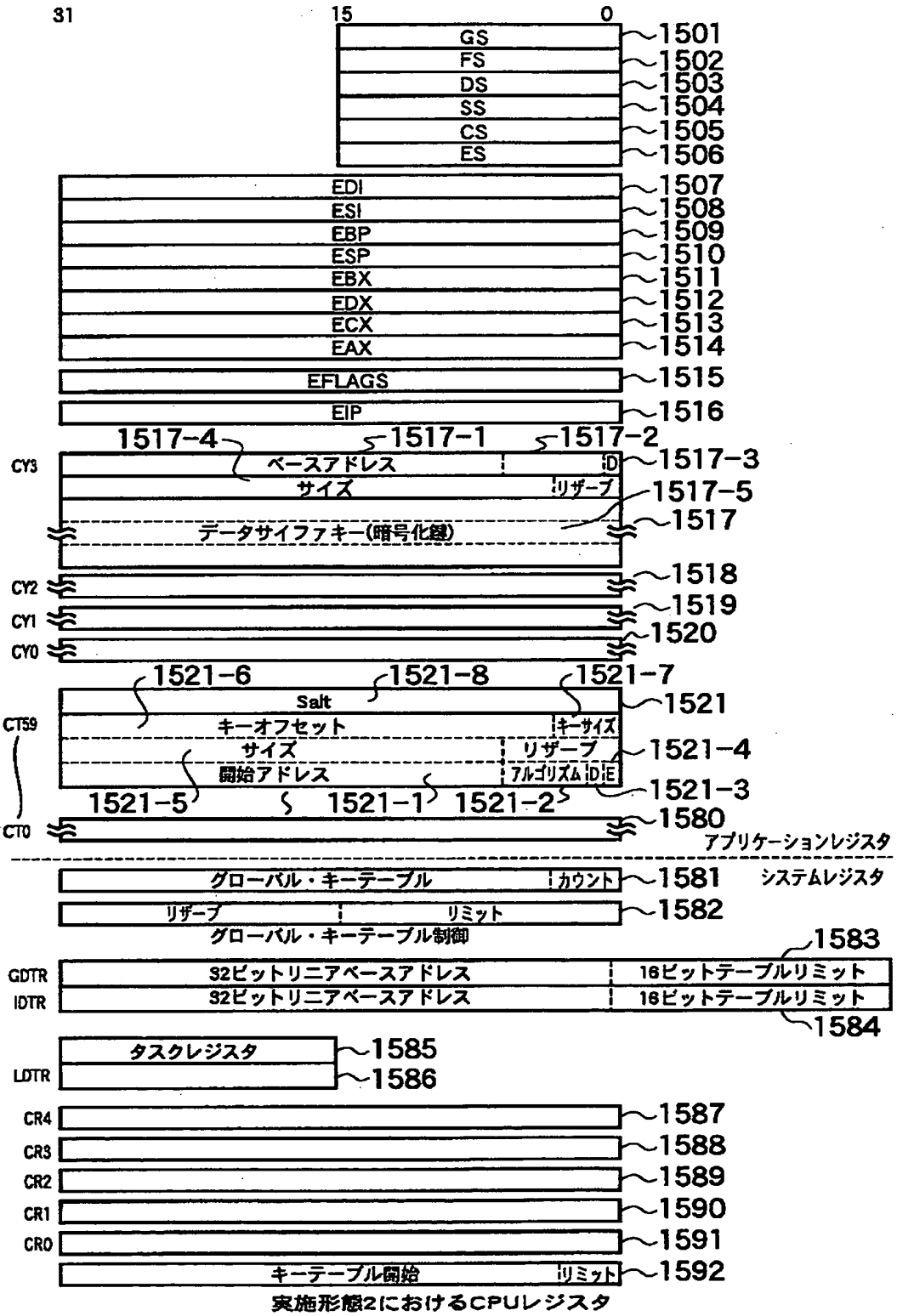
データ保護属性共有手順

【図１４】

スレッドID	ユーザTSS	パラメータ	作業中フラグ
0	uTSS0	NULL	0
TSS1	uTSS0	pblock1	0
TSS2	uTSS2	pblock2	0
⋮	⋮	⋮	⋮

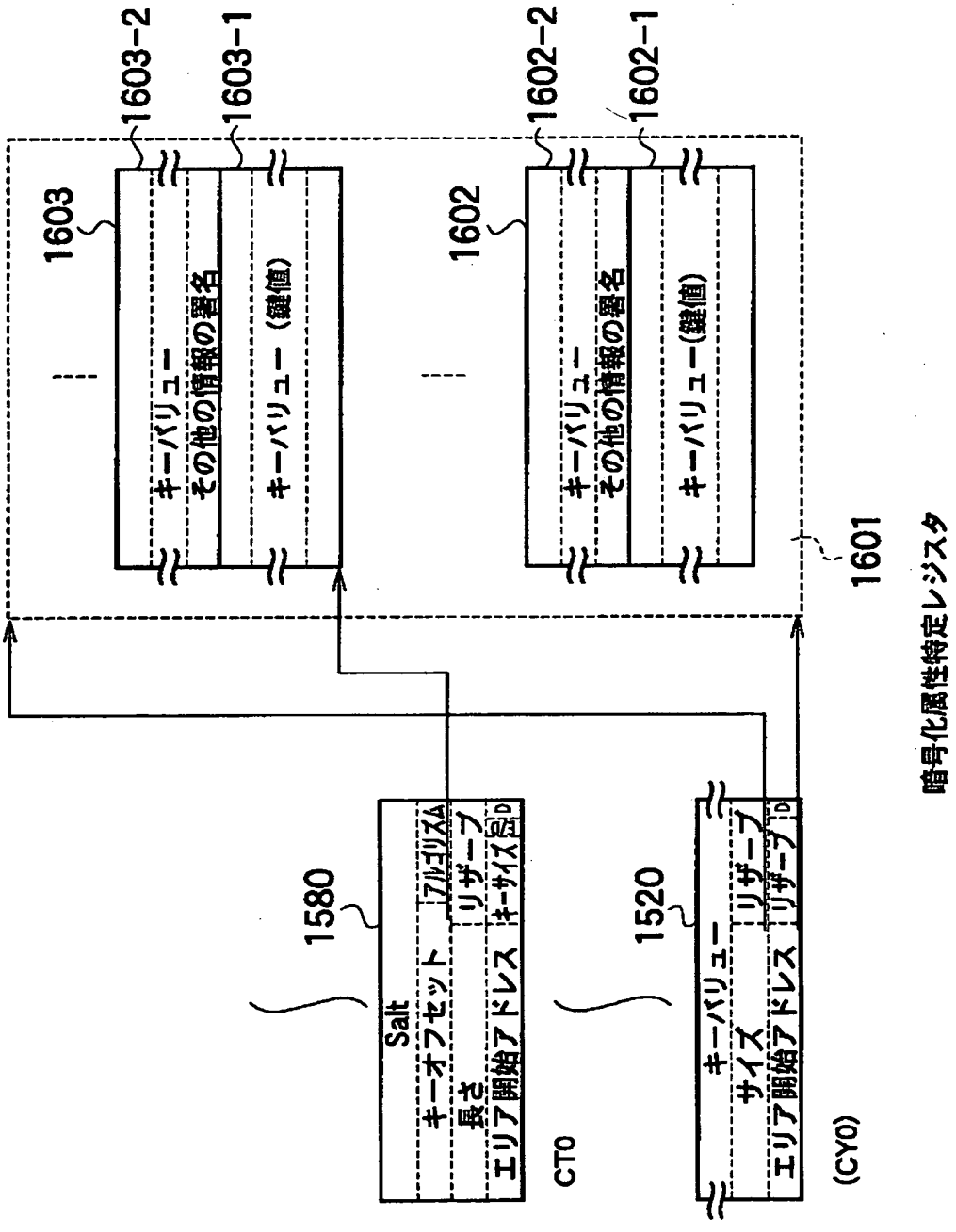
スレッドテーブル

【図１５】

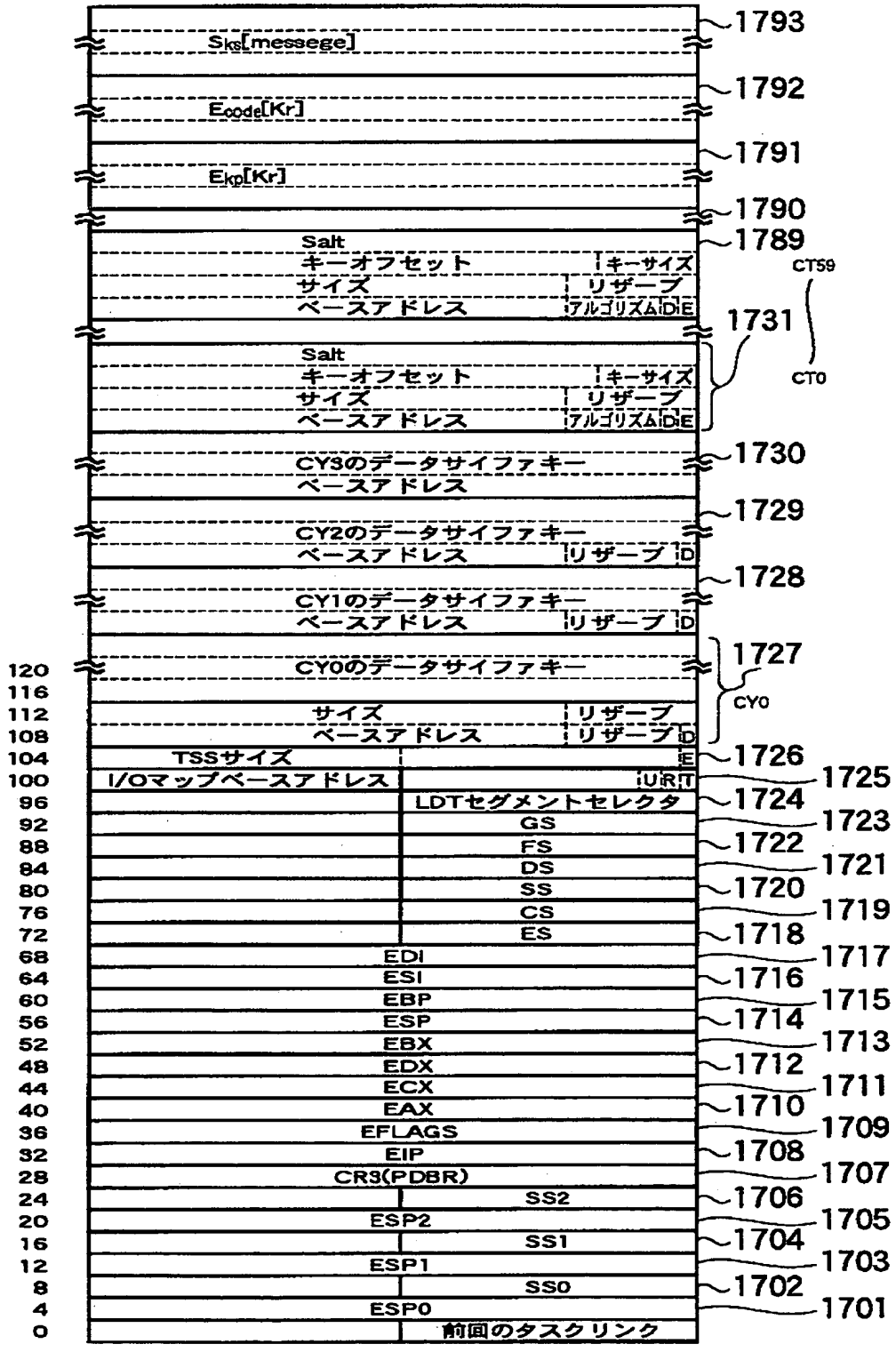


実施形態2におけるCPUレジスタ

【図１６】


1603-2
1603-1
1602-2
1602-1
1603
1602
キーバリュー
その他の情報の署名
キーバリュー（鍵値）
キーバリュー(鍵値)
1601
暗号化属性特定レジスタ
1580
1520
Salt
キーオフセット
長さ
エリア開始アドレス
リザーブ
キーサイズ
CT0
キーバリュー
サイズ
エリア開始アドレス
リザーブ
(CY0)

【図１７】



実施形態2におけるコンテキスト保存形式

【図１８】

31	15	0	
I/Oマップベースアドレス		T	100
	LDRセグメントセレクタ		96
	GS		92
	FS		88
	DS		84
	SS		80
	CS		76
	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3(PDBR)			28
	SS2		24
ESP2			20
	SS1		16
ESP1			12
	SS0		8
ESP0			4
	前回のタスクリンク		0

従来技術におけるコンテキスト保存形式

【書類名】　　要約書

【要約】

【課題】　マルチタスク環境下で、暗号化されたプログラムと、そのプログラムが扱うデータの双方を不正な解析から確実に保護することのできるマイクロプロセッサ。

【解決手段】　マイクロプロセッサは、外部のメモリから、それぞれ異なる暗号化鍵により暗号化された複数のプログラムを読み出す。コード復号化ユニットは、読み出したプログラムを対応する復号化鍵で復号化し、命令実行部が復号化されたプログラムを実行する。あるプログラムの実行が中断される場合に、コンテキスト情報暗号／複号化ユニットは、中断されるプログラムのそれまでの実行状態を示す情報と、このプログラムのコード暗号化鍵Ｋcodeとを、マイクロプロセッサに固有の公開鍵Ｋｐで暗号化し、コンテキスト情報としてメインメモリに書き込む。プログラムを再開する場合は、コード暗号化鍵・署名検証ユニットは、マイクロプロセッサの秘密鍵Ｋｓでコンテキスト情報を復号化し、復号化されたコンテキスト情報に含まれたプログラムの暗号化鍵が、このプログラム本来の暗号化鍵Ｋcodeと一致するかどうかを検証する。一致した場合にのみ、プログラムの実行再開を許可する。

【選択図】　　図２

出願人履歴情報

識別番号　〔０００００３０７８〕

１．変更年月日　１９９０年　８月２２日
［変更理由］　新規登録
住　所　神奈川県川崎市幸区堀川町７２番地
氏　名　株式会社東芝